UNIVERSITA' DEGLI STUDI
DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

BULL ITALIA
CENTRO STUDI
INFORMATICA
E AUTOMAZIONE

Note di Software

ARCHIVIO
STORICO
ASSOCIAZIONE
POZZO DI MIELE
RPDM-124
RNSW-132

Anno 1990
47
Marzo

Note di software è una pubblicazione trimestrale, edita a cura del Dipartimento di Scienze dell' Informazione dell' Università di Milano e dal Centro Studi Informatica e Automazione della Bull Italia.

Note di software intende costituire uno strumento per la circolazione di informazioni, idee, metodologie ed esperienze nell'area delle tecnologie e della produzione del software.

La collaborazione a **Note di software** è libera. Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Direttore responsabile o con il Comitato di Redazione. In particolare si sollecitano contributi nella forma di monografie, da inviare in triplice copia, e che non dovrebbero superare le cinquemila parole.
I lavori ricevuti per la pubblicazione verranno revisionati dai membri del Comitato di Revisione Scientifica, che possono avvalersi della collaborazione di specialisti del settore. I lavori pubblicati riflettono comunque le opinioni dei loro autori.

Note di software viene distribuito ad Aziende, Enti e specialisti del settore che ne facciano richiesta alla Redazione.

Direttore Responsabile: Franco Filippazzi
Bull Italia, Centro Studi Informatica e Automazione, via Vida 11 - 20127 Milano

Comitato di Redazione: Stefania Bandini, Francesco Gardin, Roberto Polillo
Università di Milano, Dipartimento di Scienze dell' Informazione
via Moretto da Brescia, 9 - 20133 Milano, tel. (02) 7575233

Comitato di Revisione Scientifica: Alberto Bertoni, Gianluigi Castelli, Giovanni Degli Antoni, Giorgio De Michelis, Mario Italiani, Gaetano Aurelio Lanzarone, Giancarlo Martella, Marco Maiocchi, Giancarlo Mauri.

Autorizzazione del Tribunale di
Milano n. 87 del 23 febbraio 1977

Supporto tecnico - gestionale e distribuzione: Domenico Asinelli
Bull Italia, via Tazzoli 6
20154 Milano, Tel. 02 - 67792960

FPDM-124

UNIVERSITA' DEGLI STUDI
DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

Bull Italia
CENTRO STUDI
INFORMATICA
E AUTOMAZIONE

QUESTO NUMERO 1

- CONSTRAINT NETWORKS
  Francesco Ricci 3

- REPRESENTING PROLOG PROGRAMS AS NETS
  Fabio Casablanca 15

- RAPPRESENTAZIONE COGNITIVA DI STRUTTURE TEMPORIZZATE
  Antonio Camurri, Agostino Poggi 29

- LE CARATTERISTICHE DEI PRINCIPALI MODELLI SEMANTICI DI DATI
  C. Tesauro 41

- S.I.S.M.A.: UN ELABORATORE SISTOLICO RICONFIGURABILE A RUNTIME
  Giuseppe Rossano, Roberto Gabaglio, Daniela Ghiroldi, Roberto Grande, Giulio Barbaglia. 55

AVVENIMENTI 61

RECENSIONI 73

Note di Software

Anno 1990
47
Marzo

QUESTO NUMERO

In diverse importanti aree della Intelligenza Artificiale si presenta il problema della "Constraint Satisfaction". Nel primo articolo Francesco Ricci introduce questa nozione e ne illustra le potenzialità applicative.

Nel secondo articolo Fabio Casablanca propone un modello di rappresentazione del Prolog in termini di teoria delle reti. Il modello (denominato Backtracked Prolog Nets) viene confrontato con gli approcci tradizionali.

Successivamente, Antonio Camurri e Agostino Poggi presentano Action Nets, un modello per la rappresentazione di azioni nel mondo reale. Esso integra metodologie di Intelligenza Artificiale, quali reti sematiche ad eredità multipla, con altre basate sulla simulazione.

Il quarto lavoro, di C. Tesauro, esamina le caratteristiche dei principali modelli semantici di dati, mettendo in luce gli aspetti di differenzazione più rilevanti.

L'ultimo articolo, infine, presenta un elaboratore sistolico riconfigurabile a runtime (SISMA). Gli autori (G. Rossano, R. Gabaglio, D. Ghiroldi, R. Grande G. Battaglia) delineano le caratteristiche del progetto e descrivono il prototipo hardware e l'ambiente di sviluppo software realizzati.

Chiudono il numero le consuete rubriche di Avvenimenti e Recensioni.

LA REDAZIONE

CONSTRAINT NETWORKS*

Francesco Ricci
Istituto per la Ricerca Tecnologica e Scientifica
Povo (TN)

ABSTRACT

In this paper the notion of a Constraint Satisfaction Problem is introduced and the main related problems are reviewed. Some examples, from several different fields of application, are used to motivate the research issues which are then illustrated. To a great extent we devote ourselves to preprocessing techniques, and in particular we state necessary conditions on the preprocessing, in order to obtain backtrack free solutions. We also review the main general heuristics for variable reordering as well as for value selection. We conclude by indicating some of the current research directions.

RIASSUNTO

In questo lavoro viene introdotta la nozione di "Constraint Satisfaction Problem" e vengono prese in rassegna le principali problematiche correlate. Vengono usati degli esempi, che spaziano in diversi campi di applicazione, per motivare i temi di ricerca illustrati dopo. Si dà largo spazio alle tecniche di "preprocessing", ed in particolare vengono enunciate delle condizioni sul "preprocessing" che assicurano una ricerca senza "backtracking". Vengono anche prese in rassegna delle euristiche generali per il riordino delle variabili e per la scelta dei valori. Si conclude indicando alcune delle attuali linee di ricerca.

* I would like to deeply thank Paolo Avesani, Cesare Furlanello, Anna Perini and Alex Simpson who reviewed and critiqued early drafts of this paper. Special thanks to Oliviero Stock who encouraged this research.

1. INTRODUCTION

This paper is an introduction to the subject of constraint networks, and is motivated by forthcoming work on a general constraint language. The constraint network problem consists in finding an assignement to a set of variables which is compatible with a set of constraints that are defined between the variables. This general problem arises in many practical situations from computer vision, automated reasoning, planning and scheduling, etc., and was first investigated in the present form by U. Montanari [29] who has stated the terminology and the main problems. Here we shall review mainly the theory, whereas the applications will be used only as testcases for the techniques the theory has developed. This paper is not meant to provide a full coverage of the subject, which is rapidly growing and branching, nevertheless we hope to give at least a comprehensive account of the deterministic techniques. We shall not be concerned with stochastic algorithms which are a growing topic of interest and would deserve a whole paper [16, 17, 20, 22, 35]. In section 2 we give the basic definitions and we pose the majority of the problems we shall be concerned with in the following. Section 3 is devoted to three examples from temporal reasoning, spatial reasoning and propositional calcu-

lus. The goal is to illustrate the main algorithmic techniques used in CSP. Section 4 illustrates the cronological back-tracking depth first algorithm and shows some of its maladies. In section 5 we go through the preprocessing issue and we review all the main research done in the subject. Finally in section 6 we examine the heuristics used in solving a CSP including heuristics for the vertical and horizontal order.

2 DEFINITIONS AND PROBLEMS

In this section we shall state some definitions and we shall pose some of the problems we shall be concerned with in the following. We begin with the formal definitions of a constraint network.

Definition 1 A constraint satisfaction problem *(CSP)* is a tuple $< X_I, D_I, R_I >$, where $I = \{1,...,n\}$, and:

1. $D_I = \{D_1,..., D_n\}$ is a set of labels, and each label D_i is a set;
2. $X_I = \{X_1,...X_n\}$ is a set of variables, where each X_i takes values in D_i;
3. $R_I = \bigcup^n_{i=1} R^i$ is a graded set of relations on the X_i, where each R^i is a set of i-ary relations, also called constraints.

We shall call every pair $< X_i, D_i >$ a node and the whole CSP a network. To solve a CSP means to find an assignement of values to the variables in X such that all the relations in R are satisfied. A binary CSP is a network in which all the constraints involve only pairs of variables that is $R_I = R^2$. A binary CSP can be associated with a graph called constraint graph in which nodes represent variables and arcs connected pairs of constrained variables.

In the following we shall be concerned mainly with binary networks. There is a precise reason indeed, every network can be "dualized" in a binary network which can be used to obtain the solutions of the original one. If C is a network, the dual network C^* has a node n^i_j for every constraint R^i_j in R^i, $j = 1,...,i_k$ and the elements of the label of n^i_j are all the i-vectors that satisfy R^i_j, (i is the arity of R^i_j). The binary constraints of C^* impose that nodes of C^*, i.e. constraints of C, which share common variables of C have the same value for those variables. The constraint graph of C^* will be called the dual constraint graph of C.

Solving a CSP is not the only problem you can pose. Here are many others:

1. Does the CSP have one solution?
2. Find all solutions;
3. If the CSP does not have a solution find the minimal set of constraints that have to be weakened to get one;
4. If a CSP has a solution what are the minimal changes that have to be made if a new variable is introduced?
5. Find all values of some variables that are part of a solution;
6. Find all values participating in all solutions (a theorem);
7. Find a minimal instantiation that entails a proposition;
8. Find an optimal solution.

All the above problems are NP-complete and researchers have also been concerned with the "weak" problems associated, as will be clear in the following. Many techniques have been developed, not only based on the famous constraint propagation one which by no means denote a single algorithm, rather a general way to achieve global consistency through local computations. We shall go through some examples, which are mainly devoted to illustrate the variety of techniques that constraint propagation point to. We now try to give a classification.

Constraint inference. New constraints are inferred and added to the network, exploiting knowledge-based relations between constraints. We shall see this in an example in temporal reasoning [2], but the reader can also see [13]. In a certain way, this is related to the solution problem. In fact to solve a CSP means to find ("synthesizing" as Freuder says) the n-ary constraint which binds all the nodes. The problem here is to find k-ary constraints, where $k \leq n$ from constraints of degree less than k.

Label inference or consistency checks. The constraints are used to refine the labels, eliminating such elements of a label that cannot partecipate in any solutions. The optimal goal is to obtain a refinement that could be used to calculate a backtrack-free solution [29, 36, 14, 15, 25, 27, 28, 19].

Value propagation. Values are assigned to variables as assumptions or as initial conditions. The constraints are exploited to find other values. These values can be entailed by the previous ones as in [34, 8, 26] or may only be possible values as happens in qualitative simula-

tion or causal models [6, 24, 12].

Another thing is worth saying. Many real-world problems have an intrinsic dynamicity, i.e. the structure of the problem change with time. Local propagation techniques seem to better tackle this important issue. Moreover finding a solution can be very expensive, from a computational point of view, also in quite simple problems. But if no local dependency is maintained, having a first solution would be of no use if we had to search a new solution for a slightly different problem. From this point comes the motivation to use a truth maintenance system, even if a great deal of extra work has to be done in mantaining the solution [5]. Finally, learning is also an issue which is studied in this context by many researchers.

3. EXAMPLES

3.1 Temporal reasoning

In this section we shall give an example of what we have previously called constraint inference. What follows is loosy drawn from the work of Allen [2] but the reader is warmly adviced to read the original paper if he is interested in Allen's treatment of time. Let $D_I = \{D_1, \ldots, D_n\}$ a set of subsets of IR, the set of real numbers. D_i represents the set of possible start times for action X_i. If X is an action we shall denote with X- and X+ the start time and the end time of X respectively, so we have X_i- $= X_i$, that is we use only one symbol to denote the action and the start time of that action. Let's suppose that the duration of every X_i is known, on this set of nodes we can define a set R of binary relations which belong to the following general classes:

a) $X \leq Y$ iff $X+ \leq Y-$
b) $X = Y$ iff $(X- = Y-)$ and $(X+ = Y+)$
c) X *overlaps* Y iff $(X- \leq Y-)$ and $(X+ \geq Y-)$ and $(X+ \leq Y+)$
d) X *during* Y iff $(X- \geq Y-)$ and $(X+ \leq Y+)$

We could have defined other types of relation but now we are not concerned with the expressivity of the relation language, which is obviously of fundamental importance in effective temporal planners. Let $C = < X_I, D_I, R_I >$ the CSP defined as above. A solution for such a network is a scheduling of the activities in X_I which satisfies the temporal constraints in R. But we can do something more, that is to say constraint inference. Let us consider another CSP C^* where $X^*_I = \{X^*_1, \ldots, X^*_m\}$ ranges over the set of pair of nodes of C. Every X*i represent an arc between two nodes of C and the corresponding label D^*_i contains all the temporal relations of C between the two nodes. The only relation in R^* is the ternary transitivity relation. For instance, if X_i *during* X_j and $X_j \leq X_k$ are known the transitivity relation states that X_j must be $\leq X_k$ (note, C^* is something more than the dual of C, the transitivity relation is "knowledge" about time).

This fact points to a general facet of CSPs. Some constraints are induced by others. We may also talk of explicit and implicit constraints. It is obvious to say that implicit constraints are redundant, they will not change the solutions set, and we shall say that two nets are equivalent if they represent the same n-ary relation. But the difficulty searching for a solution in equivalent nets can vary a lot. For instance, let $D_1 = D_2 = D_3 = \{a,b,c\}$ and $R_{12} = R_{23}$ be the strict lexicographic ordering relation. Searching for a solution by instantiating the variables in the order (X_1, X_3, X_2) takes much more effort not knowing that R_{13} is also in the strict lexicographic ordering than knowing it. Another aspect suggested by this example is that it is useful, and in some cases necessary, to have some knowledge about the constraints, about how the constraints are related, how one constraint may influence another. This knowledge is comparable to the metaknowledge in production systems [7].

3.2 Understanding line drawings

The following example is devoted to illustrate a situation in which label inference is commonly used and has originated. In this example [37] we suppose that we are living in a crack-free polyedra world with light arranged to eliminate all shadows. From the two dimensional image of a polyedra (see figure 1) we want recognize the original object, that means to identify which edges are boundary, which convex and which concave (see fig 4). Lines in a two dimensional picture can be subdivided into boundary lines (←,→), interior lines which can further subdivided into convex lines (+) and concave (-) ones. Lines conjoin in four possible ways: L; arrow; T and fork (see figure 2).

By physical constraints, lines can be labeled in a junction in such a way that only 18 of the 208 possible junctions can occur. The 18 junction configurations are di-

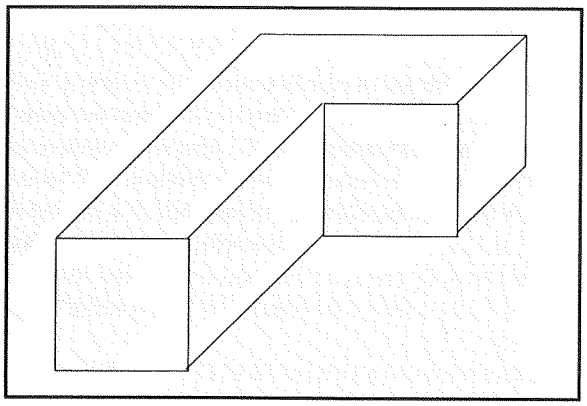

Fig. 1 A polyedra without shadows

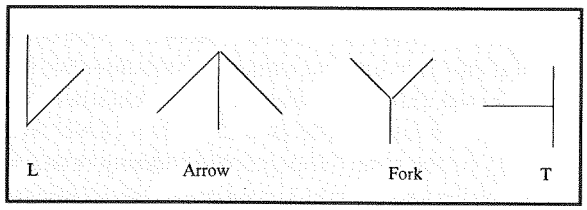

Fig.2 Types of junctions without labelling of lines

splayed in figure 3.
We can associate every picture with a binary CSP in the following way. The constraint graph is identical to the graph defined by the picture. Every vertex in the picture corresponds to a node in the constraint graph which has the set of physically possible junctions as labels. Every line in the picture corresponds to an arc in the constraint graph which is a binary constraint stating that lines connecting two vertices have the same line label on both vertices. Before trying to solve the CSP defined by a picture, searching a correct assignement for the value of every junction, one can try to restrict a label eliminating those values which cannot partecipate in any solutions. Surely a value in the label of a vertice cannot partecipate in a solution if in any connected vertex there is not a value label that has the same line label for corresponding lines. Refining the labels exploiting this principle is what Waltz has done [36] and has been called arc consistency. We shall return in the following to this subject in more depth.

3.3 Value inference in TMS

The TMS algorithm of McAllester [26] is based on "propositional constraint propagation" which was ori-

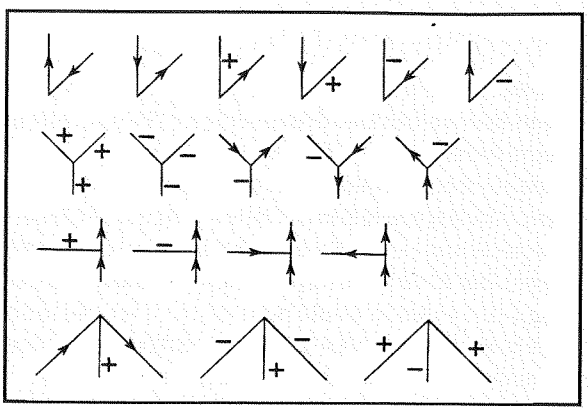

Fig. 3 The 18 possible junctions

ginally described, in essence, by Davis and Putnam [9]. This technique is related to the algebraic constraint propagation of Sussman [34] (see also [5], for a comparison between TMS and CSP). In a propositional constraint propagation system a node represents a proposition and the set of labels for that node is {*true*, *false*, *unknown*}. The constraints represent the logical relations that interweave between assertions. All logical relations take the form of disjunctive clauses such as (V(*p.false*)(*q.true*)) (read -*p* V *q*). This constraint says that every labeling of *p* and *q* is possible with the exception of *p* labeled true and *q* labeled *false*. All the logical relations (*or*, *and*,→,not) can be defined conjuncting the appropriate disjunctions. For.instance:

and (V((*and* p *q*).*true*)(*p.false*)(*q.false*))
(V((*and p q*).*false*)(*p.true*))
(V((*and p q*).*false*)(*q.true*)

This is a ternary relation between the nodes (*and p q*), *p*, *q*. The constraint states the possible consistent labeling of the nodes. Finding a consistent labeling of all nodes (assertions) installed in the TMS is the main goal that one can have when one views the TMS as a CSP. A solution from a CSP point of view is an "extension" from a non monotonic logic point of view. Historically the TMS research has pursued another way.A TMS deduces only truth values that can be inferred from a single constraint and the truth values of the nodes linked in that constraint. This process is iterated until the constraints set is relaxed. It is worthwile to stress that not all deductions which logically follow from a set of constraints are deduced by this algorithm. For instance, *p* follows from the two constraints (V(*p.true*)(*q.true*)) and

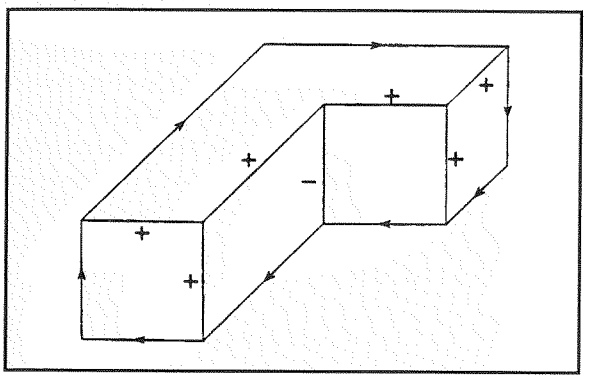

Fig.4 The polyedra of figure 1 labelled correctly

($\forall(p.true)(q.false)$) but it is not inferred by the algorithm. This is a theorem (the value *true* for p will participate in any solutions), and as we pointed out in the introduction this is an NP-complete problem. Another observation is that in this example the CSP is modified dynamically as new nodes (assertions) are put in the data base.

4. BACKTRACK AND DEAD ENDS

4.1 Backtrack

In the following section we shall be concerned with the solution problem. The basic algorithm for solving a CSP is the well known chronological backtracking depth first one. Let $< X_I, D_I, R_I >$ be a CSP and $(X_1,...,X_n)$ an ordering for the variables in X_I. Let $(x_1,...,x_i)$ an assignment for the first i variables which is compatible with the constraints in R. The following defines the basic backtrack algorithm:

Forward $(x_1,...,x_i)$

begin
1) if $i = n$ return
2) C_{i+1} ← compute the candidates $(x_1,...,x_i, X_{i+1})$
3) if C_{i+1} is not empty
4) x_{i+1} ← the first element in C_{i+1}
5) remove x_{i+1} from C_{i+1}
6) Forward$(x_1,...,x_{i+1})$
7) else
8) *Go - back*$(x_1,...,x_{i-1})$
end

Go-back $(x_1,...,x_i)$

begin
1) if $i = 0$ return. No solution.
2) if C_i is not empty
3) x_i ← first element from C_i
4) remove x_i from C_i
5) *Forward* $(x_1,...,x_i)$
6)else
7) remove the value of X_i
8) *Go - back*$(x_1,...,x_{i-1})$
end

Candidates are computed refining the labels in such a way that all the elements are compatible with the values of all the variables instantiated before.

4.2 Why backtrack fails

First of all we may notice that the search space of a CSP with n nodes and medium branching factor b, i.e, the average cardinality of the labels, is b^n. So we have exponential growth with respect to the number of nodes. Now we are going to illustrate some situations where chronological backtracking seems too naive, not only from a "human" point of view.
We are doing this in order to motivate some improvements of the basic algorithm, that greatly enhance its performance.
We start with an example. Let X_1, X_2, be two nodes with labels $D_1 = \{a, b, c, d\}$, $D_2 = \{c\}$. Let the relation between X_2 and X_1, R_{21}, be the strict lexicographic ordering, that is $R_{21}(val(X_2),val(X_1))$ if $val(X_2) < val(X_1)$, where $val(X)$ is the value of the variable X (in the following we shall not distinguish between the symbol and the value). Me may distinguish "vertical order", the order in which variables are chosen for instantiation, and "horizontal order", the order in which values are tested for a given variable [14]. If we search a solution for this CSP with vertical ordering (X_1, X_2) and horizontal ordering (a,b, c,d) for D_1, letting $val(X_1)$ = a we shall soon come to a dead end because the only value in D_2 is c and obviously R_{21} is not satisfied. The algorithm will reproduce ananalogous dead end three times, until $X_1 = d$ and $X_2 = c$.
We may make two observations. First of all if we had ordered the variables (vertical ordering) in the other possible way, i.e. (X_2, X_1) we would have found a solution without backtracking.
The other observation is that looking at the label D_1 o-

ne can say that the values $\{a,b,c\}$ have no values in D_2 that are in relation R_{21} with. Therefore, before starting brute search, one can filter these values from the label. Now suppose again that the vertical order is (X_2, X_1) but we have also ordered the label D_1 in decreasing lexicographic ordering (d,c,b,a). In that case too we would have found a solution without backtacking. Now consider another example. Let $D_1 = D_2 = D_3 = \{a,b\}$ be the labels of X_1, X_2, X_3 respectively. Let $R1_2$ and R_{23} be the equality relations and R_{13} be the complementary relation, i.e. $R_{13}(x,y)$ is true iff $x \neq y$. This CSP has no solution as a complete search demonstrates. But is it possible to discover this without explicitly doing search?

5. PREPROCESSING

A large number of dead ends the backtrack algorithm finds are due, as we have seen in the previous section, to local inconsistency. Local inconsistency can be detected before starting search and can be removed with relatively small efforts. We shall see in this chapter advantages and limitations of the preprocessing techniques. Moreover, every consistency algorithm that we will examine in the following subsection can be viewed as an instantiation of a general relaxation algorithm scheme. This scheme has been proposed by Montanari and Rossi [30], who also study its properties and characterize the classes of networks having relaxation rules of bounded complexity.

5.1 Arc consistency

As we had already seen in the previous examples, if $C = < X_I, D_I, R_I >$ is $\in$ CSP, a $\in D_i$ and there is no b $\in D_j$ such that $R_{ij}(a,b)$, where R_{ij} is the intersection 1 relation between nodes i and j, then we can remove a from D_i, because it will not participate in any solutions. If this does not occur, that is, if for all a $\in D_i$ there exists a $b \in D_j$ such that $R_{ij}(a,b)$, and vice-versa, then we shall say that the arc (i,j) is consistent. If every arc in C is consistent we shall say that C is arc consistent. This simple observation is at the basis of arc consistency algorithms [36, 25, 27, 28, 19]. These algorithms basically iterate label refinement in every node until nothing else changes. We shall not go deeply into these algorithms. What we want to stress here is that arc consistency can be achieved in polynomial time but it is not sufficient for global consistency. After arc consistency is achieved there may still be some values in a label which do not participate in any solutions. For instance, let $I = \{1,2,3\}, D_1 = D_2 = D_3 = \{a,b\}$ and $R_{12} = R_{13} = R_{23}$ be the "not equal" relation. This CSP is arc consistent but it has no solutions.

5.2 Path consistency

Inconsistency is generated when constraints are not explicit. Let go back to the previous example, if X_1 = a and $X_2 = b$ then there is no value in D_3 which can be assigned to X_3 which is not equal to a and b. Here a ternary constraint which states the mutual inequality of the variables is only implicitly stated by the binary constraints. Let $C = < X_1, D_1, R_1 >$ be a CSP. If for every triple (i,j,k) and for every pair (a_i, a_j), where $a_i \in D_i$, $a_j \in D_j$ and $R_{ij}(a_i,a_j)$, there exists a value $a_k \in D_k$ that closes the path, i.e. such that $R_{ik}(a_i,a_k)R_{ik}(a_j,a_k)$, then we say that C is path consistent [29]. As for arc consistency, in the literature there are some algorithms which achieve path consistency in polynomial time [29, 25, 27, 28, 19].
What we want to stress is that path consistency is only a necessary condition for global consistency. We slighty modify the previous example (the graph colouring problem). Let $C = < X_I, D_I, R_I >$ a CSP where $I = \{1,2,3,4\}, D_i = \{a,b,c\}$ for $i = 1,2,3,4$. Every node has a link with every other nodes and the relation is "not equal". This network is arc and path consistent but it has no solutions.

5.3 High order consistency

Freuder [13, 14] has generalized these two notions with the concept of k-consistency. Let C be a CSP, we shall say that C is k-consistent if for every choice of k - 1 variables along with values for each that satisfy all the constraints among them, for any kth variable there exists a value for this kth variable such that the k values taken together satisfy all the constraints among the k variables. We shall also say that C is strong k-consistent if it is j-consistent for all $j \leq k$.
Freuder introduces another notion; the width of an ordered constraint network. If $C = < X_I, D_I, R_I >$ is a network of binary relations and $(i_1,...,i_n)$ is an ordering of I, the width of this ordered constraint network is the maxinium width at the nodes, where the width at the node

Relations have an obvious graded algebra structure with union and intersection operations

X_{ij} is the number of relations in R_i of the type R_{ijik}, where $k < j$. Freuder proves the following in [14]:

Theorem 1 (Freuder): Given a constraint satisfaction problem, a vertical search order is backtrackfree if the level of strong consistency is greater than the width of the corresponding ordered constraint graph.

So one is led to conclude that the wider a network is the stronger the consistency necessary to get a solution, or to decide that a solution does not exist. This is nearly right. In fact, suppose the constraint graph of *C* is a complete graph and every link represents the equality relation. In this case after only arc-consistency you can find a backtrack-free solution irrespect of the width of this constraint graph which is n - 1, where n is the number of nodes.
Let's go back again to the graph colouring problem, when: $I = n$, $\# D_i = n$-1 and the graph is complete, i.e. every node is linked with every other node. The width of this network does not depend on the ordering and it is n - 1. This CSP has no global solution because we need n colours for such a graph and we need the strong n-consistency to realize it. Strong (n-1)-consistency is not enough.
This example shows that in general a n-width constraint network cannot be solved with lower levels of consistency.
Looking at theorem 1 the program for solving a CSP would be, first to find out a ninimum width ordering, second to achieve the required k-consistency and finally to get a backtrack-free solution. Regarding the first point the obvious way to follow is a branch-and-bound process. Repeatedly expand a node at the end of a minimal width branch by adding as its children the variables not yet used in that branch until a branch using all the variables is completed.
Regarding the second point an important observation has to be made, following Freuder himself. The algorithm that Freuder adopts to achieve k-consistency may alter the connectivity structure; and the width, of the problem [14, 11]. In fact this algorithm makes explicit constraints previously only implied by other constraints. Therefore the procedure must be repeated until the width does not change any more. Let us give a simple example that illustrates this aspect. Let *C* be the graph colouring problem with a complete graph of 4 elements and only 3 colours, all links represent the "not-equal" relation. Let *C'* be another CSP equivalent to *C* but where X_4 is linked to X_2 and X_3, and X_5 is linked to X_1, X_5 is also linked to X_4 with an equality relation (see figure 5). *C'* has width 2 but is not 3-consistent. Achieving 3-consistency will create exactly the implicit link X_4 X_1 [13]. In conclusion it may happen that no advantage can be taken of the fact that a CSP possesses a width-2 constraint graph if it is not already path consistent.

5.4 Consistency for backtrack-bounded search

In a subsequent paper Freuder [15] gives a more general notion of width and consistency. The final goal is to establish a sufficient condition for a backtrack-bounded search, that is a search where backtrack goes back at most a predetermined number of levels. The width of a group of *k* consecutive vertices in an ordered constraint graph is the number of vertices preceding the group in the ordering that are linked to any of the k vertices in the group. The *j-width at* a vertex v is the minimum, for

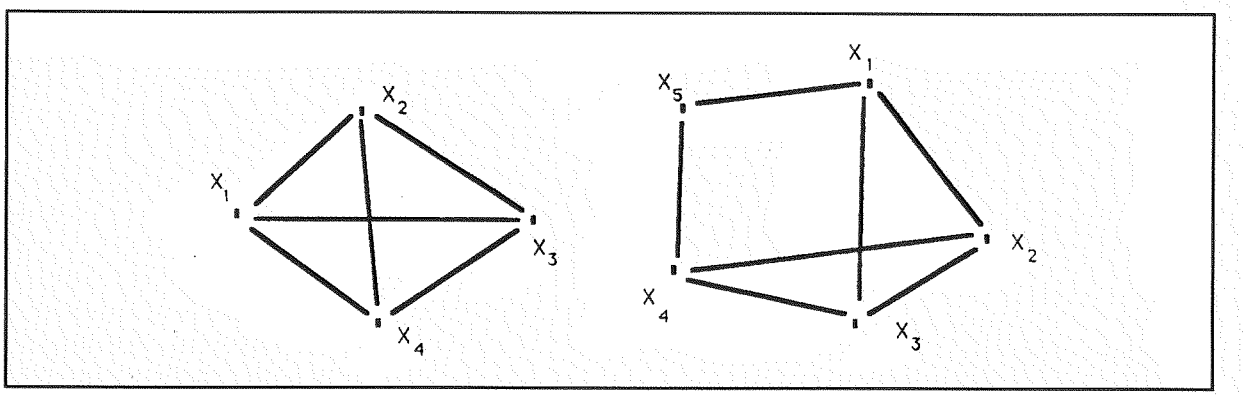

Fig.5 two equivalent network

$k = 1,...,j$ of the width of k consecutive vertices up to and including v. The j-width of an ordered constraint graph is the maximum j-width at all vertices in the ordering. The j-width of a constraint graph is the minimum j-width of all orderings of the constraint graph. The following generalize the old notion of k-consistency. We shall say that a constraint graph is (i,j)-consistent if given values for i variables, satisfying the constraints on those variables and given any other j variables, it will be possible to find values for those j variables such that the $i + j$ values taken together satisfy all constraints on the $i + j$ variables. The following generalizes theorem 1:

Theorem 2: *Given a constraint graph for a constraint satisfaction problem, there exists a vertical search order that guarantees k-bounded backtrack search if the graph is strongly (i,k)-consistent for i equal to the k-width of the graph.*

The program for solving a CSP with theorem 2 in mind would be the same as for theorem 1. First achieving a j-width ordering then the required (j,k)-consistency, where k is the bound in backtrack that we want. But unfortunately the same difficulties occur.

5.5 Directional and adaptive consistency

Dechter and Pearl [10] have given a weaker definition of k-consistency, namely directional k-consistency, which is both sufficient for guaranteeing backtrack-free solutions and adds in general fewer arcs to the constraint graph, preserving the graph width in a larger class of problems. Here is the definition for directional path-consistency, the general case is analogous.

Definition 2: *A constraint graph C is d-path-consistent w.r.t ordering* $d = (X_1,..., X_n)$ *if for every pairs of values* (x,y), $x \in D_i$, $y \in D_j$ *such that* $R_{ij}(x,y)$ *and for every* $k > i,j$, *there exists a value* $z \in D_k$ *such that* $R_{ik}(x, z)$ *and* $R_{kj}(z,y)$.

The following theorem is the analogous of theorem 1 and it is proved quite straight-forwardly in [10]:

Theorem 3: *Let d be a width-2 ordering of a constraint graph C. If C is directional arc and path-consistent w.r.t d, then C is backtrack-free.*

But here again applying directional path-consistency, which is $O(n^3b^3)$, to a width-2 graph may increase its width and therefore, does not guarantee backtrack-free solutions. Fortunately one can characterize those constraint graphs that do not increase in width after d-path-consistency has run. This is what Arnborg has done [3]. In the same paper a further weakening of the notion of consistency is stated. Given an ordered constraint graph in order to instantiate a variable X_i according to all the constraints, limitations can be produced only by the n_i variables that precede X_i in that particular order and that are linked to X_i by some constraint. In other words the quantity of checks depends on the variable. Adaptive consistency processes an ordered constraint graph in decreasing order achieving step by step the required degree of consistency.

Adaptive-consistency $(X_1,...,X_n)$

1. begin
2. for $r = n$ to 1 by -1 do
3. compute $PARENTS(X_r)$
4. perform Consistency(X_r, $PARENTS(X_r)$)
5. connect by arcs all elements in $PARENTS(X_r)$ (if they are not yet connected)
6. end

Where the procedure Consistency(V,SET) records a constraint among the variables in SET, induced by V. The following theorem is derived easily.

Theorem 4: *An ordered constraint graph processed by Adaptive-consistency is backtrack-free.*

5.6 Tree clustering

Theorem 1 is really of practical interest when the constraint graph is a tree. In this case the corresponding CSP can be solved efficiently in $O(nb^2)$ steps, i.e. after arc consistency is achieved a backtrack-free solution is at hand, instantiating the variables in any width-1 order. Dechter and Pearl [11] take advantage of this observation and give a procedure to form clusters of variables such that the interactions between the clusters are tree-structured, and then they solve the problem by efficient tree algorithms. This approach seems very appealing as it is based on the general strategy "divide and conquer" (see also [23]). A point in the tree-clustering algorithm seems doubtful. The clusters the algorithm produces

correspond to maximal clique (complete subgraph) of the chordal primal graph (a graph is chordal if every cycle of length at least four has a chord). But is well known that deciding if a graph contains a k-clique is an NP-complete problem [1] and therefore also finding all maximal cliques.

5.7 Consistency during search

When the backtrack algorithm runs, the values of the variables are changed dynamically. As new variables become bound new restrictions are imposed on the other variables. Looking ahead, partial looking ahead and forward checking [18] are three techniques which exploit this simple fact, namely when the procedure forward is called these procedures do a test on the variables which have not been instantiated yet. This test exploits the knowledge at hand in that point of the search, and if it fails go-back is called in anticipation.
The procedure forward in section 4.1 is called with the variables $(X_1,...X_i)$ bound to $(x_1,...x_i)$ and with the variables $(X_{i+1},...,X_n)$ not instantiated yet. We shall call $(X_1,...,X_i)$ the bound variables and $(X_{i+1},...,X_n)$ the free variables.
The looking ahead procedure tests if each free variable has at least one value which is compatible with the values currently held by the bound variables, and if each free variable has at least one value which is compatible with one of the possible values for each other free variable.
Partial looking ahead is a simplification of looking ahead. Again one tests that each free variable has at least a value which is compatible with the values assigned to the bound variables but in this case is checked only that each free variable has at least a value which is compatible with some value only of the sequent free variables in the given order.
Forward checking is the simplest checking technique. The test is done only on each free variable determining if it has at least one value which is permitted by the current assignment of the bound variables. Haralick and Elliot give a complete experimental account of this technique applied to the N-Queens problem and to random generated problems. Savings in backtrack are compared with the additional consistency checks required. Quite surprisingly the simplest one, that is forward checking has the highest mark.

6. HEURISTICS

Preprocessing techniques try to simplify the search for a solution in CSP, eliminating values which will not participate in any solutions. This gives rise to an equivalent CSP which is simpler to solve. However, as we have seen, this doesn't solve all problems. In some cases the resources spent "perfectly preprocessing" a CSP, that is in achieving the required degree of consistency, may equate with the resources that one should have spent solving directly the CSP. But even worse. sometimes preprocessing is waste of time, that is the time required to understand the necessary degree of consistency plus the time required to achieve it, may exceed the time spent by pure backtracking. So we must face search, but how we can drive search in order to minimize the waste of resources? The answer is "heuristics". In this paper we shall use the word "heuristics" in a broad sense. The way for improving search in the backtrack algorithm is twofold. First, when and how to go ahead, second when and how to go back. If we are searching a solution for a CSP and some variables have been already instantiated, satisfying all the constraints, we should face two types of choices: which variable to chose and which value to try first, that is which value is more likely to lead to a consistent solution. Further, when a dead end is encountered, or when it is likely a dead end will be encountered, which assigned values it is better to change? In the following sections we shall investigate the previous questions.

6.1 Choosing the variables

As we have seen in the previous examples the order in which the variables are chosen for instantiation (the so called "vertical order") may be extremely important in reducing the number of backtracks. But in general is not clear what is the best vertical order and how this can be found out.
Bitner and Reingold [4] proposed a general search techniques that advances the search along the most constrained path. That is, when faced with the choice of several ways of extending the partial solution, we choose the variable that offers the fewest alternatives. Stone and Stone [33] used this technique in a classical testbed, the N-Queens Problem, that is the problem to put on an

n-chessboard n queens in such a way that no queen attacks each other. They compared the lexicographic search, i.e. the default way in which the backtrack algorithm works, with the most-constrained search for n up to 29. Quite surprisingly in this problem the number of backtracks in the most-constrained search has un upper bound aproximately at one hundred, whereas in the lexicographic search the number of backtracks grows exponentially with the number of queens, and for 29 queens millions of backracks are required to find the first solution.
These empirical results seem quite convincing but it is not clear how far they can be extended to the general case and what are the precise reasons this behaviour is produced. In a more theoretical study Purdon [31] uses random generated conjunctive normal form predicates to test the most-constrained search, also called search rearrangement, in a more general situation. There are three final remarks that Purdon makes:

> There is a significant class of problems where most-constrained search uses polynomial average time, while ordinary backtracking uses exponential average time.
>
> For most of the "difficult region" for ordinary backtracking, most-constrained search also takes exponential time. Most-constrained search converts only a small fraction of the region from exponential to polinomial.
>
> For most places outside of the difficult region there is no reason to use most-constrained search (indeed, because it uses more time per node one should avoid using it). The difficult region, however, contains problems most often solved by backtrack.

These remarks invite us to use most-constrained search with some moderation. Most-constrained search doesn't yeld to polynomial solution for most of the backtrack exponentially time consuming problems, but in some cases it does and for the difficult problems reduces the time needed in finding the first solution.

6.2 Choosing the value

Haven instantiated k variables to minimize the use of backtracking we should try first the values which are more "likely" to lead to a consistent solution. In the previous section we had seen the simple but efficient principle for choosing the vertical order, now we shall be concerned with the horizontal order. It is worth noting that the choice of the variable is not independent from the choice of the value to assign. Actually a general characteristic of heuristic reasoning is that decisions at a particular step are made on the basis of the expected outcome of some further steps. The point to hold in mind is that an heuristically informed horizontal order intuitively rates each value with the inverse of the probability of being part of a consistent solution of the CSP. The problem is how to gauge these probabilities.
We shall assign to each pair (variable, value) a non-negative number which depends on the other variables. We shall call this number the support, so if X_i and X_j are variables and $v \in D_i$, $S_{ij}(v)$ is the support for assigning v to X_i from X_j. In general $S_{ij}(v)$ depends on the value and on the label of X_j. If the value of X_j is known $S_{ij}(v)$ is simply the truth value of the relation between X_i and X_j. But if the value of X_j is not known, and this is the interesting case, we would have preferably defined something like $S_{ij}(v/L)$, where L is a set of possible values for X_j (a subset of D_j).
Having defined this local support we can also define a global support for the value v to X_i with, $S_i = \Pi_{i \neq j} S_{ij}(v)$ (the choice of the product to integrate local contribution is arbitrary, the only condition that needs to be satisfied is that $S_i(v) = 0$ if some $S_{ij}(v) = 0$) (see also [21]).
As an example of S_{ij} functions, $S_{ij}(v)$ could be the number of values $w \in D_j$ s.t. $R_{ij}(v,w)$. Another way of computing the S_i functions has been proposed by Dechter and Pearl [11]. If the backtrack algorithm has instantiated the first i variables, for each of the remaining variable all the values that are not consistent with the i previously instantiated variables are removed. Then a precomputed tree approximation of the constraint graph made of the n - i remaining variables is used and $S_{ij}(v)$ is set to the number of solutions consistent with the value v for X_i in the approximated problem.

7. CONCLUSIONS

In this paper we had defined the Constraint Satisfaction Problem and we have shown how it arises in some clas-

sical fields of Artificial Intelligence. We have also shown some serious maladies of the backtracking algorithm, which is the basic search technique used for solving a CSP. This has motivated the introduction of a set of techniques which are ultimately concerned with the problem of reducing the time spent in searching for a solution. Even if the general problem is NP-complete, to a certain extent these approaches can improve the behaviour of the backtracking algorithm in a considerable number of cases. A great deal of work is still to be done in classifying these "simple" problems [10, 30]. We believe that new interesting perspectives will be opened by these investigations as well as by the algebric approaches [32] and those aimed to translate CSP into TMS [5].

8. REFERENCES

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, THE DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS, Addison-Wesley, Reading, 1974.

[2] James F. Allen. MAINTAINING KNOWLEDGE ABOUT TEMPORAL INTERVALS, 26(11), 832-843, 1983.

[3] S. Arnborg, EFFICIENT ALGORITHMS FOR COMBINATORIAL PROBLEMS ON GRAPHS WITH BOUNDED DECOMPOSABILITY - A SURVEY, BIT 25, 2-23, 1985.

[4] J.R. Bitner and E.M. Reingold, BACKTRACKING PROGRAMMING TECHNIQUES, Comm. ACM, 18, 651-656, 1975.

[5] Johan de Kleer, A COMPARISON OF THE ATMS AND CSP TECHNIQUES, In Proceedings of the Eleventhth International Joint Conference on Artificial Intelligence, pages 200-296, 1989.

[6] Johan de Kleer and J.S. Brown, THEORIES OF CASUAL OREDRING, Artificial Intelligence, 29, 1986.

[7] Johan de Kleer, Jon Doyle, Guy L. Steele, jr., and Gerald Jay Sussman, AMORD: EXPLICIT CONTROL OF REASONING, SIGART Newsletter, (64), 116-125, August 1977.

[8] Jon Doyle, A TRUTH MAINTENANCE SYSTEM, Artificial Intelligence, 12, 231-272, 1979.

[9]Martin Davis and Hilary Putnam, A COMPUTING PROCEDURE FOR QUANTIFICATION THEORY, Journal of the ACM, 7:201-215, 1960.

[10] Rina Dechter and Judea Pearl, NETWORK-BASED HEURISTICS FOR CONTRAINTS SATISFACTION PROBLEMS, Artificial Intelligence, 34, 1-38, 1987.

[11] Rina Dechter and Judea Pearl, TREE CLUSTERING FOR CONSTRAINT NETWORK, Artificial Intelligence, 38, 353-366, 1989.

[12] Kenneth D. Forbus, QUALITATIVE PROCESS THEORY, Artificial Intelligence, 24, 85-168, 1984.

[13] Eugene C. Freuder, SYNTHESIZING CONSTRAINT EXPRESSIONS, Communication of ACM, 21, 958-966, 1978.

[14] Eugene C. Freuder, A SUFFICIENT CONDITION OF BACKTRACK-FREE SEARCH, Journal of the ACM, 29(1):24-32, 1982.

[15] Eugene C. Freuder, A SUFFICIENT CONDITION OF BACKTRACK-BOUNDED SEARCH, Journal of the ACM, 32(4), 755-761, 1985.

[16] Y. P. S. Foo and Y. Takefuji, STOCHASTIC NEURAL NETWORKS FOR SOLVING JOB-SHOP SCHEDULING: PART 1. PROBLEM REPRESENTATION, in Proceedings of the IEEE second international conference on neural networks (ICNN), volume II, pages 275-282, 1988.

[17] Y. P. S. Foo and Y. Takefuji, STOCHASTIC NEURAL NETWORKS FOR SOLVING JOB-SHOP SCHEDULING. PART 2. ARCHITECTURE AND SIMULATIONS, in Proceedings of the IEEE second international conference on neural networks (ICNN), volume II, pages 283-290, 1988.

[18] Robert M. Haralick and Gordon L. Elliot, INCRE-

ASING TREE SEARCH EFFICIENCY FOR CONSTRAINT SATISFACTION PROBLEMS, Artificial Intelligence, 14, 263-313, 1989.

[19] Ching-Chih Han and Chia-Hoang Lee, COMMENTS ON MOHR AND HENDERSON'S PATH CONSISTENCY ALGORITHM, Artificial Intelligence, 36, 125-130, 1988.

[20] Mark D. Johnston and Hans-Martin Adorf, LEARNING IN STOCHASTIC NEURAL NETWORK FOR CONSTRAINT SATISFACTION PROBLEMS, In Proc. NASA conf. on space telerobotics, 1989.

[21] Mark D. Johnston, KNOWLEDGE BASED TELESCOPE SCHEDULING, In A. Heck and F. Murtag, editors. Knowledge-Based systems in astronomy, pages 33-49. Springer-Verlag, Berlin, 1989.

[22] S. Kirkpatrik, C. Gelatt, and M. Vecchi, OPTIMIZATION BY SIMULATED ANNEALING, Science, 22, 671-680, 1983.

[23] Richard E. Korf, PLANNING AS SEARCH: A QUANTITATIVE APPROACH, Artificial Intelligence, 33, 65-88, 1987.

[24] Ben Kuipers, QUALITATIVE SIMULATION, Artificial Intelligence, 29, 289-338, 1986.

[25] Alan K. Mackworth, CONSISTENCY IN NETWORK OF RELATIONS, Artificial Intelligence, 8:99-118, 1977.

[26] David McAllester, AN OUTLOOK ON TRUTH MAINTENANCE, AIM 551, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1980.

[27] Alan K. Mackworth and Eugene C. Freuder, THE COMPLEXITY OF SOME POLINOMIAL NETWORK CONSISTENCY ALGORITHMS FOR CONSTRAINT SATISFACTION PROBLEMS, Artificial Intelligence, 26, 65-74, 1985.

[28] R. Mohr and T.C. Henderson, ARC AND PATH CONSISTENCY REVISITED, Artificial Intelligence, 28, 225-233, 1986.

[29] Ugo Montanari, NETWORKS OF CONSTRAINTS: FUNDAMENTAL PROPERTIES AND APPLICATIONS TO PICTURE PROCESSING, Inf. Sci., 7:95-132, 1974.

[30] Ugo Montanari and Francesca Rossi, EXACT SOLUTION IN LINEAR TIME OF NETWORK OF CONSTRAINTS USING PERFECT RELAXATION, In Ronald Brachman, Hector Levesque, and Ray Reiter, editors. Proceedings of the first international conference on principles of knowledge representation and reasoning, pages 394-399, 1989.

[31] Paul Walton Purdon, jr., SEARCH REARRANGEMENT BACKTRACKING AND POLINOMIAL AVERAGE TIME, Artificial Intelligence, 21, 117-133, 1983.

[32] Igor Rivin and Ramin Zabih, AN ALGEBRAIC APPROACH TO CONSTRAINT SATISFACTION PROBLEMS, In Proceedings of the Eleventhth International Joint Conference on Artificial Intelligence, pages 284-289, 1989.

[33] Harold S. Stone and Janice M. Stone, EFFICIENT SEARCH TECHNIQUES, AN EMPIRICAL STUDY OF THE N.QUEENS PROBLEM, IBM j. res. develop., 31(4), 464-474, 1987.

[34] Gerald Jay Sussman and Guy L. Steele, jr., CONSTRAINTS: A LANGUAGE FOR EXPRESSING ALMOST-HIERARCHICAL DESCRIPTIONS, Artificial Intelligence, 14, 1-39, 1980.

[35] G.A. Tagliarini, SOLVING CONSTRAINT SATISFACTION PROBLEMS WITH NEURAL NETWORKS, In Proceedings of the IEEE first international conference on neural network (ICNN), volume III, pages 741-747, 1987.

[36] D.L. Waltz, UNDERSTANDING LINE DRAWINGS OF SCENES WITH SHADOWS, In The Psycology of computer vision, pages 19-91. McGraw-Hill, New York, 1975.

[37] Patrick Winston, ARTIFICIAL INTELLIGENCE, Addison-Wesley, Reading, 1984.

REPRESENTING PROLOG PROGRAMS AS NETS

Fabio Casablanca (*)
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

1. INTRODUCTION

Compared to other programming languages, Prolog takes advantage of a clear semantics expressed by a very compact syntactical form. However, the computational mechanism, notwithstanding the ease of use, is not always self-evident. Establishing a correspondence between a class of nets and the set of Prolog programs can be useful to show some characteristics of the procedure calls and the different levels of the informational exchange. Moreover, a language abstract machine model can be defined by nets. These models are the basis of several actual implementations of Prolog. In the field of net theory, the main reference is constituted by the Petri Nets Theory, specially suitable to describe nondeterministic behaviours. We hope that in this way the results can be used to define abstract machine models for concurrent logic programs. The developed net model is called Backtracked Prolog Nets (BP-Nets) model. The next section contains a short introduction to Prolog. After that, the BP-Nets are informally introduced, and the relation between Prolog and traditional programming languages is discussed. Then it is shown how a description by nets enlightens the role played by local and global information in Prolog. Finally, a software tool designed to translate Prolog programs into BP-Nets and simulate them is introduced.

2. PROLOG: SOME MAIN FEATURES

We assume that, Prolog being a well-known language, the reader knows its basic form. As a logic language, its fundamental characteristics are:

1. the use of procedures written as sets of logical clauses (called rules and facts), where the logical variables are the procedure parameters;
2. the triggering of a clause proof as equivalent of a program computation;
3. the exploitation of the pattern matching mechanism to choose a rule during a computation, and similarly, the exploitation of the unification mechanism to assign values to the parameters;
4. the use of backtracking mechanism to guarantee that the evaluation will look exhaustively for a successful end of the computation.

Actually, the computation rule has this form:

1. It is sequential, i.e. it uses the procedure rules in the order they are written, activating the first one whose parameters are unifiable with the calling goal variables;

(*) Presently at Toyhashi University, Japan

2. It is depth-first, i.e. it proceeds to satisfy the current goal testing first its subgoals, suspending the satisfaction of goals at its same level;
3. It exploits the backtracking, by which a same goal can have more satisfactions (it means that a goal satisfaction is not necessarily deterministic);
4. It can compell determinism by a system predicate (cut);
5. It uses system predicates for arithmetical and control functions.

In this document, we will refer, as semantical Prolog model, to the Byrd Model. This model is based on 4 fundamental operations on the predicates, which are:

1. CALL (the triggering of a predicate evaluation);
2. FAIL (the failure of a predicate evaluation);
3. EXIT (the successful end of a predicate evaluation);
4. REDO (the restarting of an evaluation, after the failure of the next predicate or after a request for new arguments values).

These four operations are the nucleus of an interpreter description as a "re-cursive cycle of pattern matching and subgoal evaluation".The main limitations we impose are:

1. excluding the control system predicates (we would have to define self-modifying nets);
2. reducing, for simplicity, the data-structures to lists.

3. AN INFORMAL INTRODUCTION TO BACKTRACKED PROLOG NETS

While we show the main features of the BP-Nets, we justify them, according to the Prolog characteristics we showed in par. 2. Mainly we want to establish a correspondence between nets and Prolog procedures. Hence a BP-Net can be built using other BP-Nets, as a procedure can be built using other procedures. In such a way, recursive nets can be defined, as recursive programs are defined, precisely using two different places:

1. name-places, just one on the same net, which gives the name to the procedure and represents the point where it is possible to enter in a lower net level or to exit in an upper net level;

2. queue-places, which represent a subnet call and a subprocedure.
In fig. 3.1. we show a graphical example of relation between nets, where name-places are white and queue-places are shadowed. The arrows specify the binding between a name-place (expanded in the direction of the right arrow) and the correspondent queue-place (contracted in the direction of the left arrow). We can have nets without queue-places, for example the subnet in fig.3.1. To make more versatile the net building procedures, the transition in this nets subclass can have an associated function, called association, which breaks the net nesting. This possibility copes with:

- the procedure development method, which can be unspecified in some parts, during the development phase;

- the need to execute non-logical (irreducible to places) functions, as mathematical and control functions.

Every net has an **environment**, which is a set of variable name/variable value couples. The idea of environment tries to solve the opposite needs to have global information, as in any programming language, and to maximize local information.
In fact, an important net feature is the possibility to go

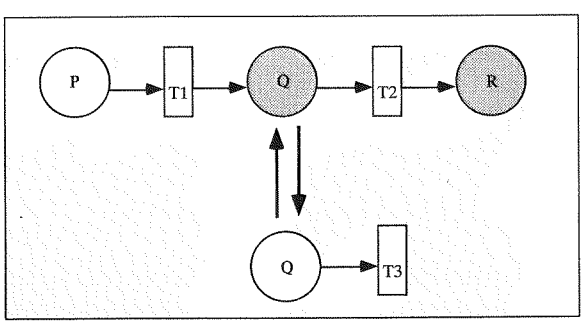

Fig. 3.1

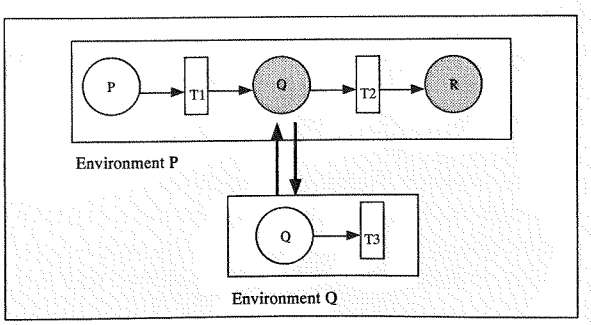

Fig. 3.2

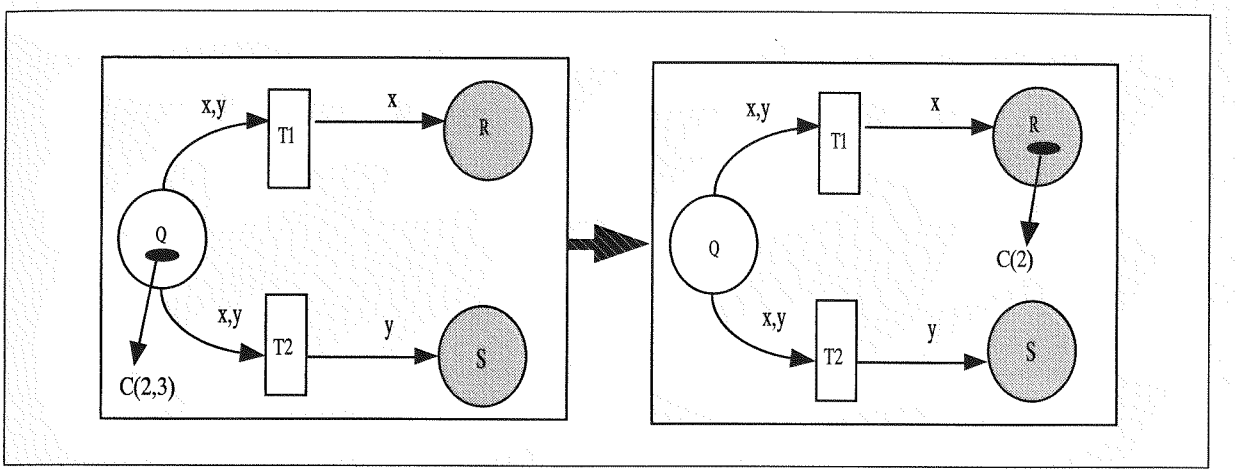

Fig. 3.3

through a net using only local information. From the Prolog side, that means to compute a program by distributed control.
In fig. 3.2. we show graphically environments and communications between environments from the fig. 3.1 situation. We call a **net firing sequence** a transition firing sequence that, starting from a token in a name-place, comes back to that name-place after going through the net.
A net firing sequence is related to a corresponding Prolog procedure. During a net firing sequence, the token gives the computation current state. More precisely, the token contains 2 colors:

1. the current operation running on the net (**0-color**, the Byrd operations corresponding to the nets operations are between parenthesis);

1.1. 'C' - triggering of firing sequence (CALL);
1.2. 'R' - retriggering of firing sequence (REDO);
1.3. 'E' - success of a firing sequence (EXIT);
1.4. 'F' - failure of a firing sequence (FAIL).

2. the currently used variables values (**P-color**).

Depending on the colors of the current token, a transition firing can have different meaning. Possibly, these values can inhibit some transition firings.

While the 0-color transformation is invariant to the transition, the P-color depends on the transition entering and exiting arcs.
The entering arc imposes constraints on the P-color values; the exiting arc gives the P-color new values.
When more transitions can fire, an **order** is supposed to exist on them; this order is the same of the sequential order of Prolog rules and facts.
The environment remembers the fired transition through a **guard** (the order function value for that transition), leading a possible next firing sequence process.
In fig. 3.3 we give an example of transition firing where this mechanism works.

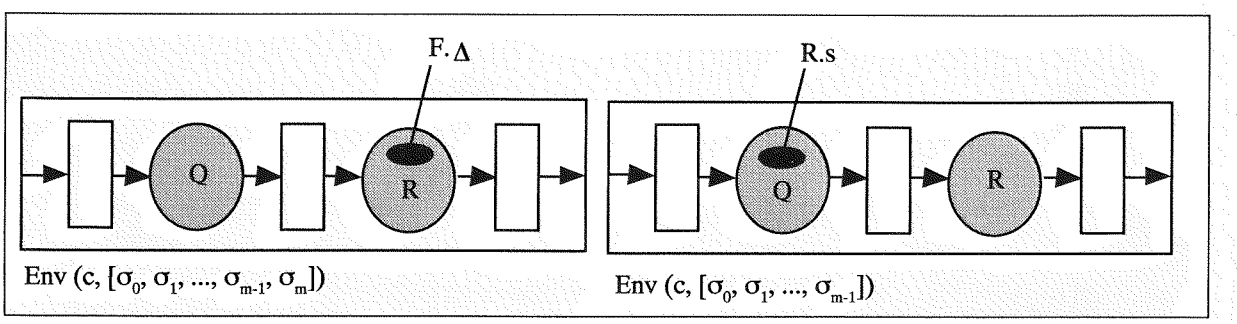

Fig. 3.4

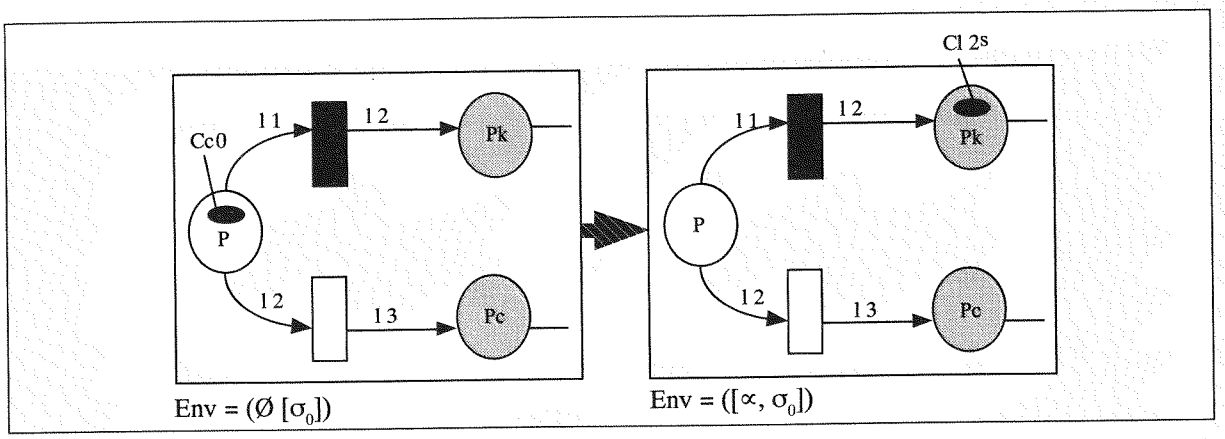

Fig. 3.5

On its net, the environment works as a stack, pushing the firing transitions constraints. These constraints are abandoned when a firing sequence partially fails, causing a backward firing transition sequence.
Finally we can introduce the last feature, the distinction between deterministic and nondeterministic transitions. Shortly, a deterministic transition binds the net firing sequence to its firing, not allowing resatisfactions with different transition firings.

In fig. 3.5., the deterministic transition (black) stops a possible next net firing sequence, imponing guard = ∝. No following firing sequence is now possible, because we will need a transition whose function order is more than infinite. In synthesis, the objects we introduced for the BP-Nets are:

1. name-places and queue-places;
2. deterministic and non-deterministic ordered transitions;
3. associations;
4. environments;
5. tokens with 0-color and P-color.

4. A TRANSITION FIRING RULE FOR BACKTRACKED PROLOG NETS

The possible situations on a BP-Net can be divided in eight big classes, obtained multiplying the types of places (2: name-places and queue-places) by the 0-color elements (4:C,F,E,R).

1. 'C'on name-place;
2. 'R'on name-place;
3. 'F' on name-place;
4. 'E' on name-place;
5. 'C' on queue-place;
6. 'R' on queue place;
7. 'F' on queue-place;
8. 'E' on queue-place.

We will consider the distinction between deterministic and nondeterministic transitions inside this class partition, when it produces meaningful differences.

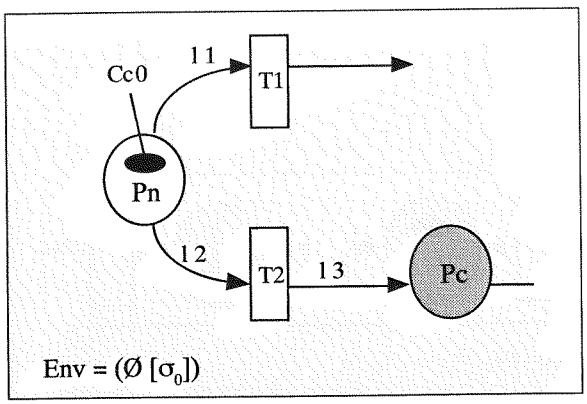

Fig. 4.1

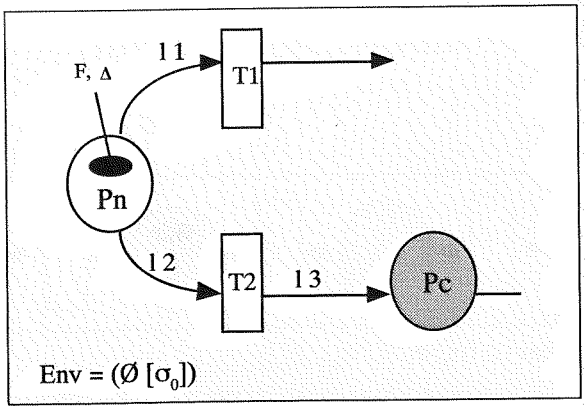

Fig. 4.2

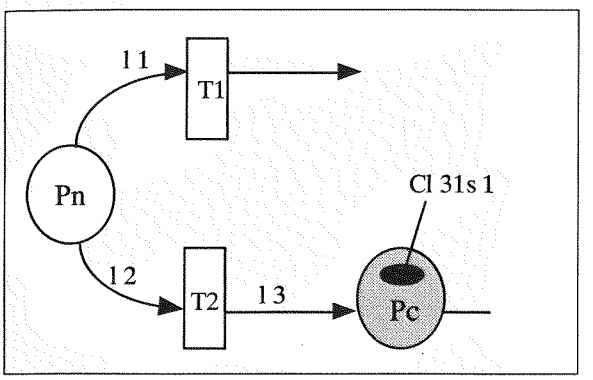

Fig. 4.3

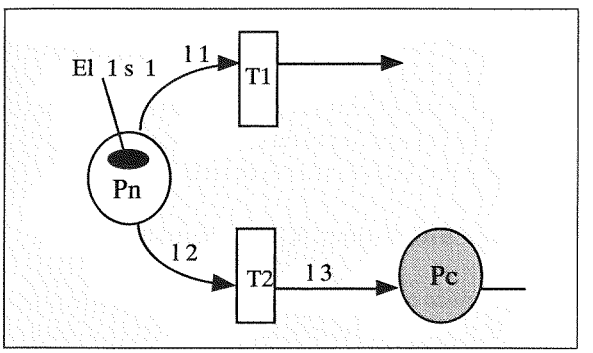

Fig. 4.4

4.1 Token with 0-color 'C' on name-place

A transition fires when:

1. the label λ_1 (linked to the arc connecting name-place and transition) and the token P-color χ_0 are unifiable;

2. no transition with higher order function value can fire. If no transition can fire, a token (F,θ) is generated on

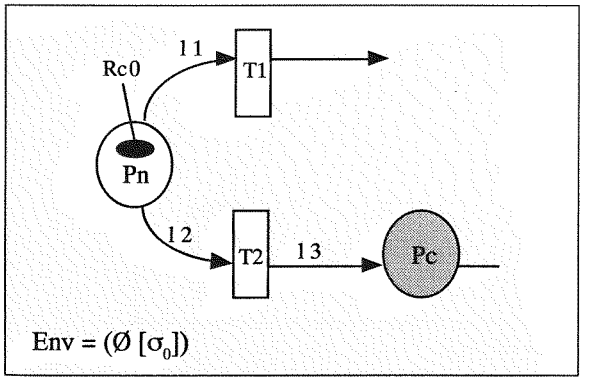

Fig. 4.5

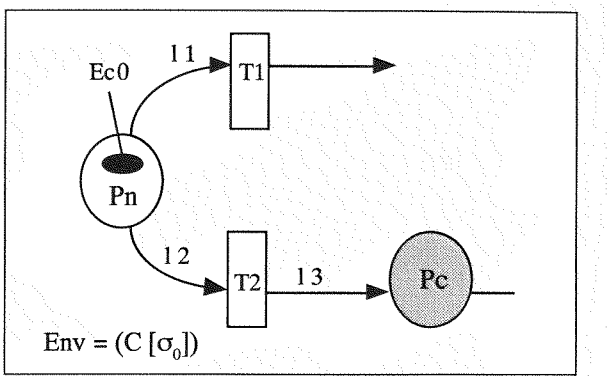

Fig. 4.6

the name-place and the environment is canceled.
Otherwise, the situation depends on the presence of queue-places.
When there is a queue place, such as the transition T_2, if σ_1 is the unifier between χ_0 and λ_2, a token (C,λ_3,σ_1) is generated on the queue-place and the environment becomes Env = $(2[\sigma_0,\sigma_1])$.
In the second case, a token (E, λ_1, σ_1) is generated on the name-place. The environment becomes E = $(1, [\sigma_0])$.
In these last two cases, if the arc is deterministic, the environment guard would have value ∝, impeding the net firing sequence. The fired transition is the only one allowed to end the net firing sequence.

4.2. Token with 0-color 'R' on name-place

This situation is very similar to the previous one, but here only transitions whose order function is superior to the guard c are allowed to fire. Transitions with order function inferior to c were already considered, with unsuccessful results.

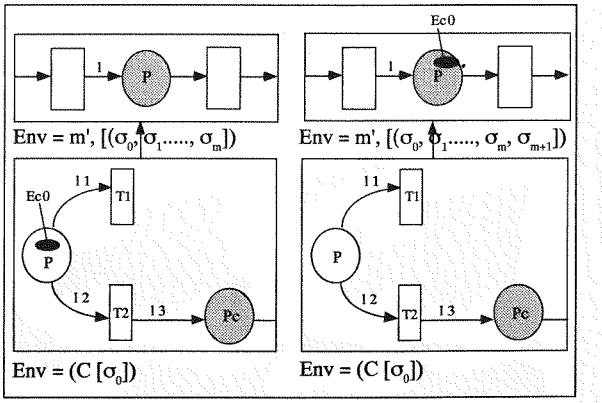

Fig. 4.7

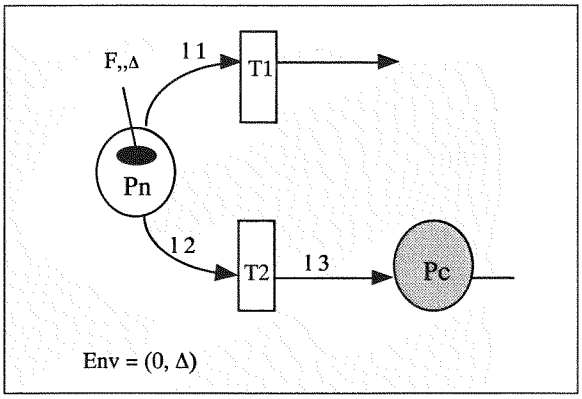

Fig. 4.8

4.3. Token with 0-color 'E' on name-place

If the concluded firing sequence is generated by the expansion of a queue-place an another net, a token (E, χ_0) is generated on this net and the environment becomes Env' = (c', [σ_0, σ_1,..., σ_m,']), where σ_m'= (v_0 / x_0, v_1 / x_1,.. v_n / x_n), if χ_0 = (x_0 / x_1, .. x_n) and σ_0 = v_0 / w_0, v_1 / w_1,.. v_n / w_n .
The environment Env is associated to the net, for possible following firing sequences.

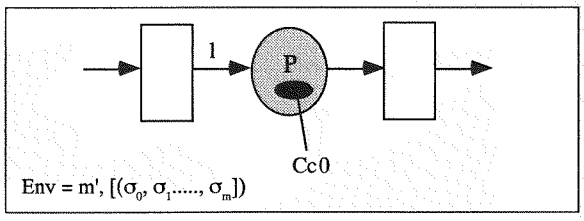

Fig. 4.10

Otherwise the process is concluded.

4.4. Token with 0-color 'F' on name-place

In this case P-color = 0 always holds, as it can be seen looking at the situations where a token with 0-color = F is generated. If the failed net firing sequence is generated by a superior level net, a token (F,O) is generated on this net and the environment is left untouched.
Otherwise no operations is executed.

4.5 Token with 0-color 'C' on queue-place

Here the place expands in the joined subnet, carrying a token (C,χ_0) and an environment Env = (0,σ), where σ' = {v_0/v_0, v_1/v_1,..., v_n/v_n}, if χ_0= (v_0, v_1,..., v_n). If this net

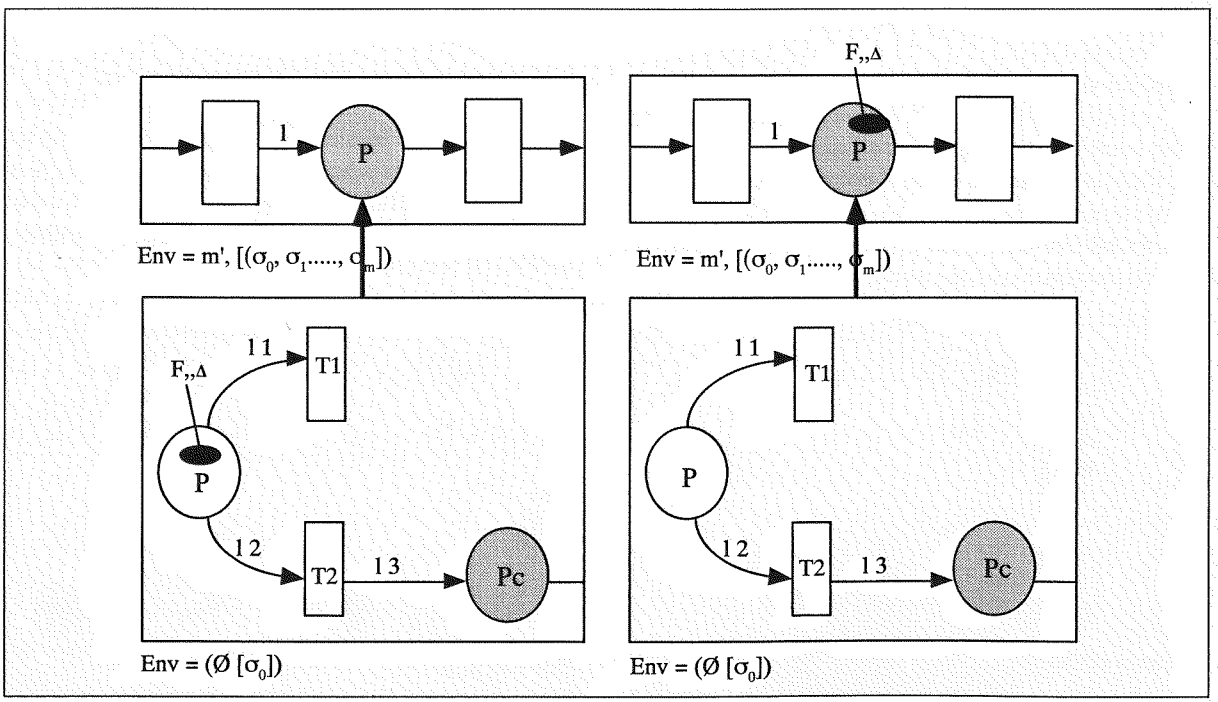

Fig. 4.9

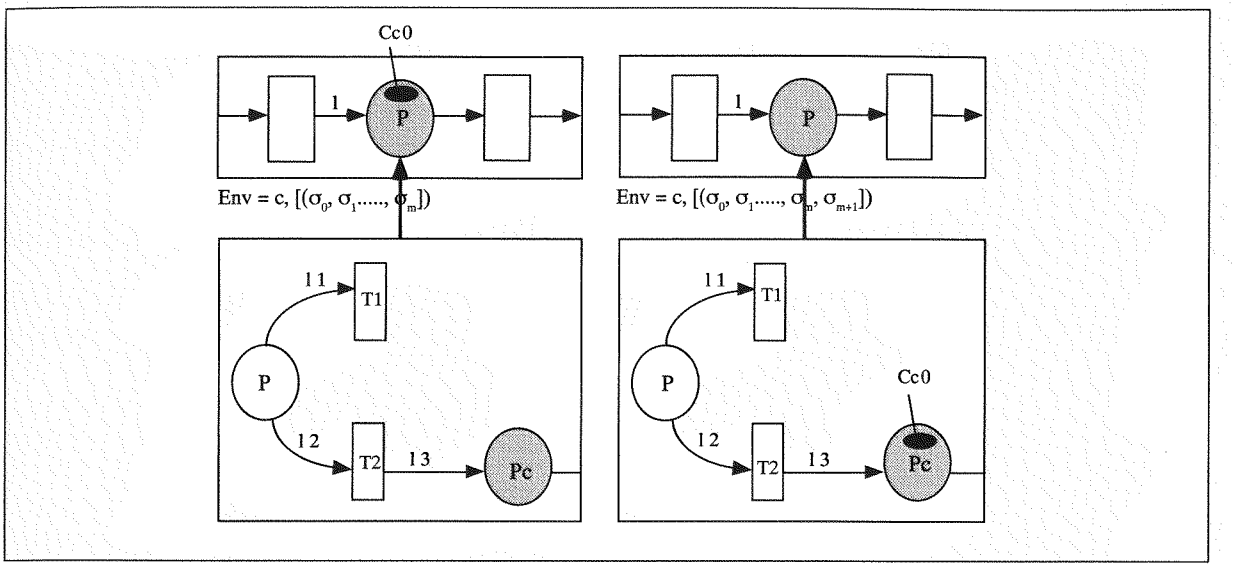

Fig. 4.11

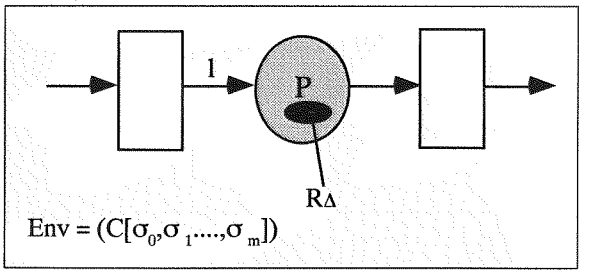

Fig. 4.12

doesn't exist, a token (F,0) is generated.

4.6. Token with 0-color 'R' on queue-place

This situation is closely analogous to the previous one. The difference lies in the existing environment of the net; the token (R, χ_0) is generated from this environment.

The net bound to the place P surely exists because a net firing sequence was already concluded.

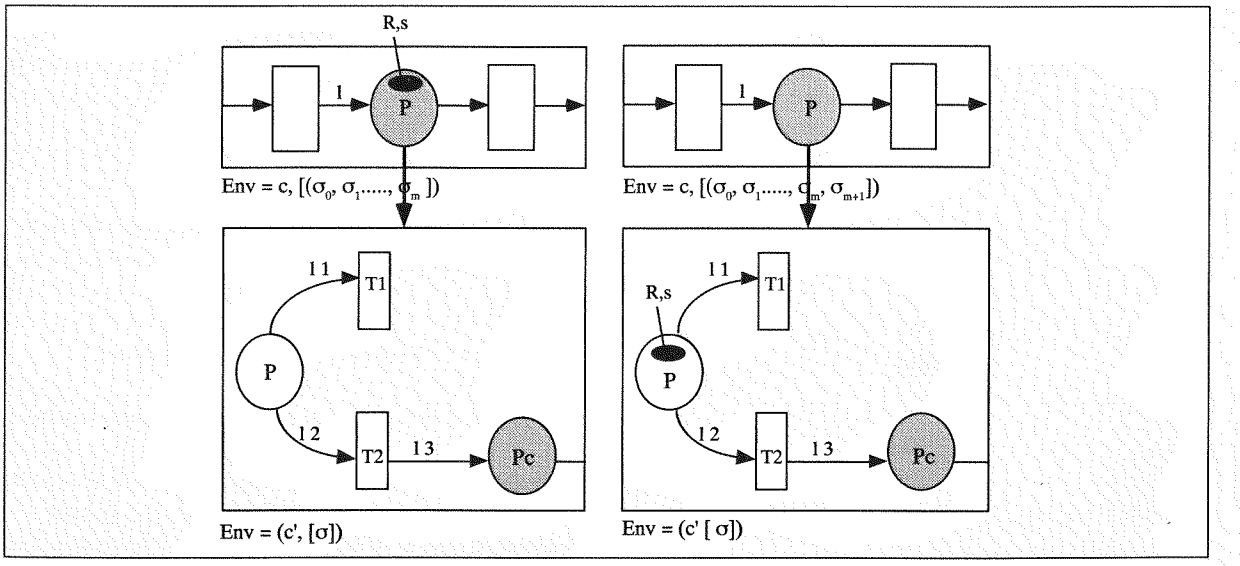

Fig. 4.13

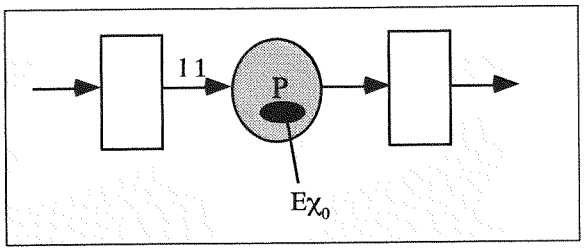

Fig. 4.14

4.7. Token with 0-color 'E' on queue-place

The situation with 0-color E is one of the most complex.

In the P-color is θ, the net is fired backwardly.
On the previous place (always existing), (E,θ) is generated to continue the backward firing sequence.
The net environment will become Env = (c, [σ_0, σ_1,..., σ_{m-1}, σ_m]), except when the place P' is a name-place. In this case Env = (c, [σ_0, $\sigma_\&$]).

If the P-color is not θ, the forward firing sequence is not concluded.
If λ_1 is the arc label going into the next place, on such a place a token M = (C, λ_1) is generated, where s = $\sigma_0 \circ \sigma_1 \circ ... \sigma_m \circ$.
If the fired transition is deterministic, it will be c ∝.

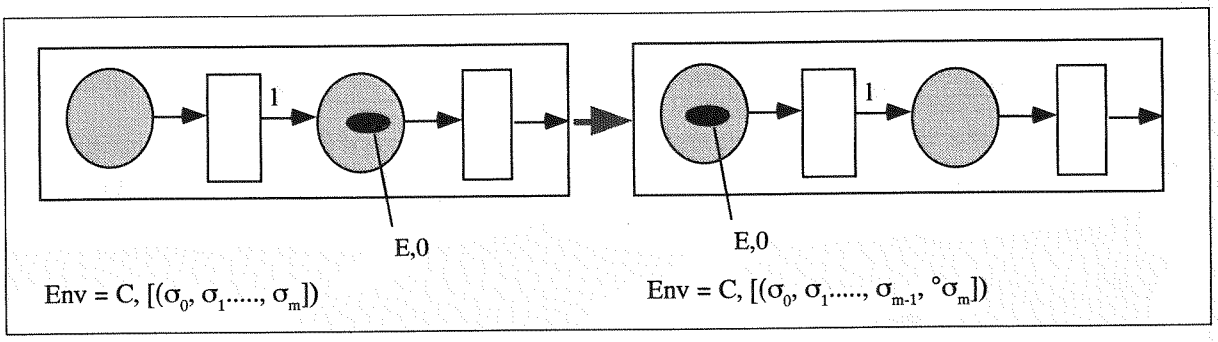

Fig. 4.15

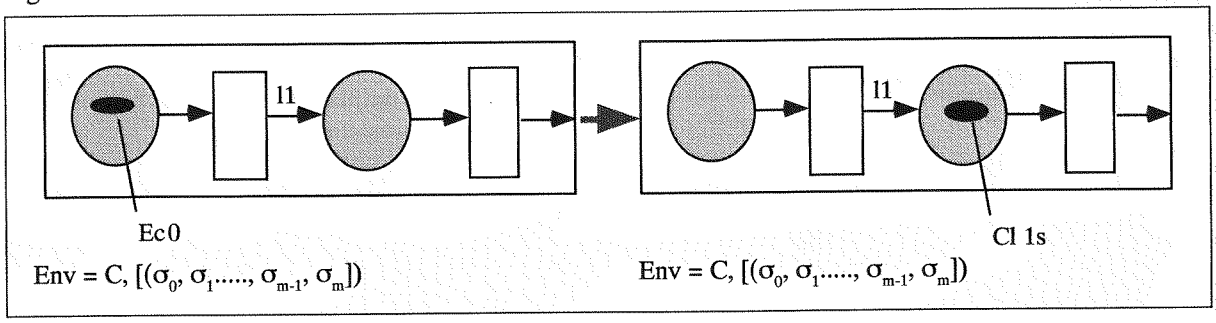

Fig. 4.16

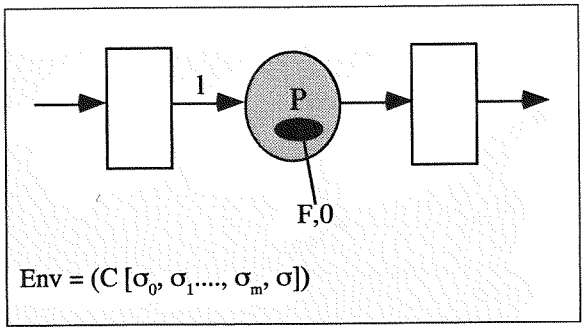

Fig. 4.17

If such a place doesn't exist, the backward firing sequence starts according the operations showed in fig. 4.15.

4.8. Token with 0-color 'F' on queue-place

In this situation a token is carried on the previous place.
If the crossed arc is deterministic, the token is M = (F,0). Otherwise the token M = (R,0) is produced and the environment Env = (C[σ_0, σ_1...., σ_{m-1}, σ_m]) becomes Env = Env = (C [σ_0, σ_1...., σ_{m-1}]).

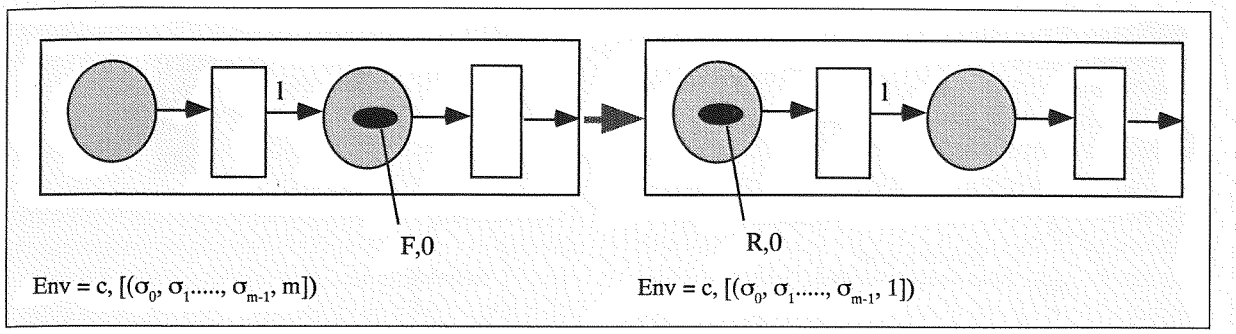

Fig. 4.18

5. THE CORRESPONDENCE BETWEEN PROLOG AND BACKTRACKED PROLOG NETS

The [DEF. 1.3.] will give a generic concept of BP-Net equivalent to a Prolog program.
The next definition will give the equivalence for a Prolog rule and, worthly, allows 'cut' on the BP-Nets. This operation is realized mapping the 'cut' predicates on deterministic transitions and the normal predicates on queue-places.

[DEF.1] If a clause Prolog $R(X_1,..., X_n)$:- $S_0(Y_{0,0},...,Y_{0,f(0)}),..., S_m(Y_{m,0},...,Y_{m,f(m)})$

1.1 it has a name - place R;
1.2 if q predicates are '!' ('cut'), it has (m+1-q) queue-places $S_{g(j)}$, where g (j)=k if and only if between S_0 and S_K there are k-j is a cut
1.3 it has (m+ 1 - q) transitions t_j and t_j is deterministic if and only if $S_{g(j)} = S_k$ and S_{k-1} is a cut;
1.4 the label λ of the arc (R, t_0) has value $(X_1,.....,X_n)$
1.5 the label λ of the arc $(t_j S_{g(j)})$ has value $(Y_1,0...,Yjk_{(g(j))})$
1.6 the label λ of the arc $(S_{g(j)} t_j)$ is equal to Ø;

is called equivalent to R.

The net equivalent to a fact is obtained too as a particular situation from this definition. For example, from the two Prolog clauses:

child(john,jane),
father(X,Y): child(X,Z), husband(Y,Z)

the nets are obtained.

[DEF. 2] If there is a sequence of n Prolog clauses R_j with same arity and name(packet) and the n BP-Nets BN_j are equivalent to them, the net BP-Net obtained reducing the BN_{js} is called equivalent to the packet.

The reduction is obtained in this way:

2.1. all the name-places of BN_js are ordered according the order of equivalent clauses and reduced to one name-place s_0;
2.2. this name-place inherits the name-places previous connections;
2.3. the transitions t_{j0}, directly connected to s_0, are ordered according to their parent net (hence as in the Prolog clauses).

Now full generality can be given.

[DEF. 3] If a Prolog program Π is given, a BP-Nets set I is equivalent to a program if every packet has an equivalent BP-Net into the I set.
The existence of an equivalent BP-net for a given packet and of an equivalent BP-Nets set can be demonstra-

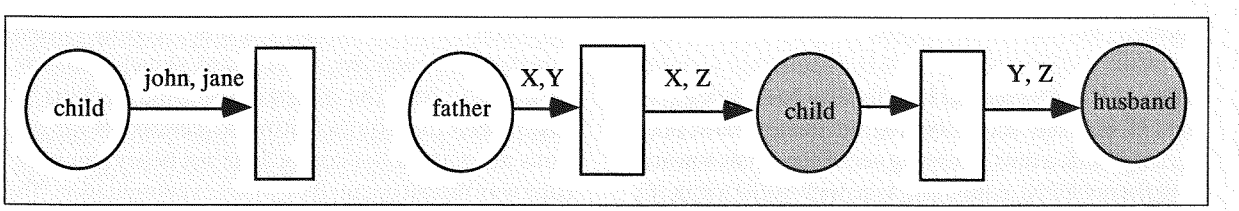

Fig. 5.1

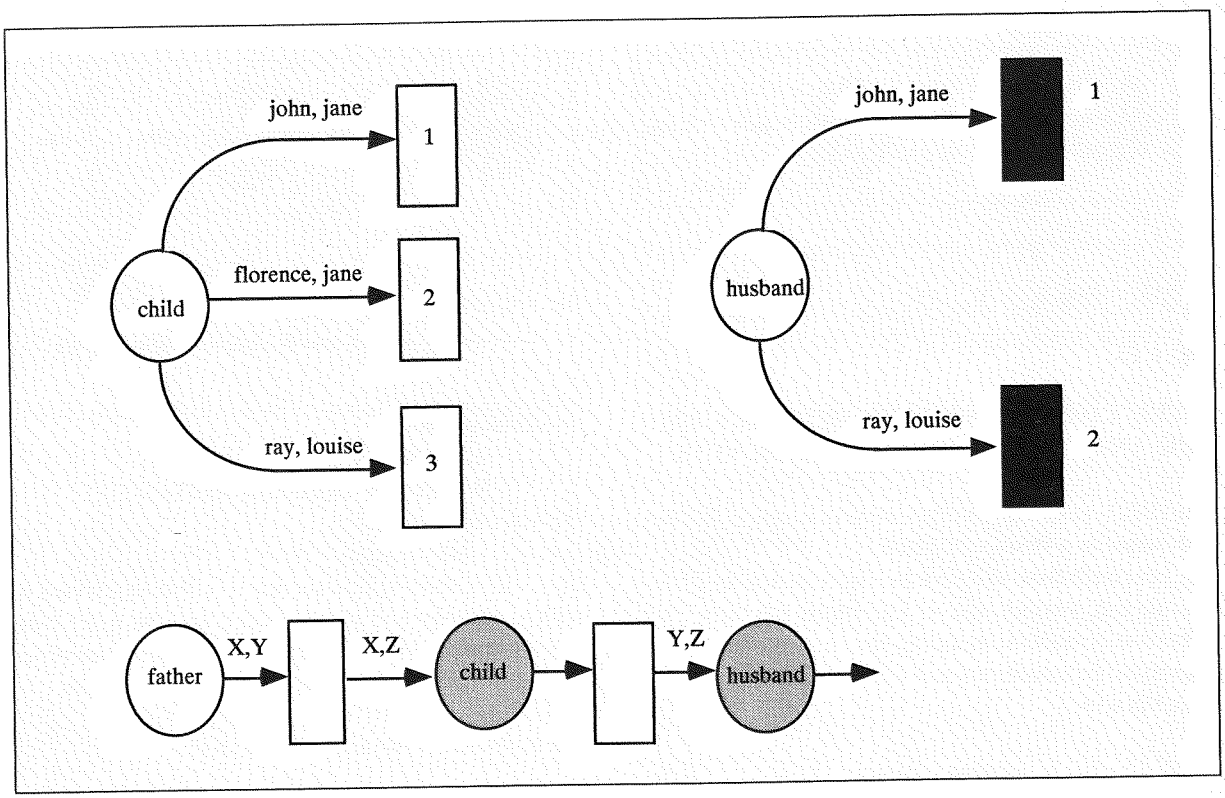

Fig. 5.2

ted by construction; because we didn't introduce a formal definition for the BP-Nets, such a demonstration is omitted. For example, let us suppose to have the Prolog program:

child(john,jane).
child(florence,jane).
child(ray,louise).
husband(carl,jane):-!
husband(eric,louise):-!.
father(X,Y):-child(X,Z),husband(Y,Z).

the equivalent BP-Net Set will be as in fig. 5.2.

These definitions give a syntactical equivalence: it is possible to prove that they induce a semantical equivalence.

[THE. 1] If a BP-Nets set I, equivalent to a Prolog program Π, is given, for every program computation $p(x_1, x_2,..., x_n)$ in Π, a nets' set firing sequence exists starting from the net BPN_p, equivalent to p, with initial token $(C, x_1, x_2,..., x_n)$. If the final value of the program parameters is $(y_1, y_2,...,y_n)$, the 0-color value, when the firing sequence is concluded, will be $(y_1, y_2,...,y_n)$. This firing sequence is called computation-equivalent.

The theorem is supporting showing how the Prolog computation rule features illustrated in par. 2. still hold on the BP-Nets. Prolog is sequential: on the BP-nets the fired transition choice respects an order, built according to the program order.
Similarly, the computation rule is depth-first. The presence of a token 'C' on a queue-place generates the expansion of the associated sub-net, and by this procedure possibly a net nesting is induced. In fact this behaviour holds on every BP-Net.

If a token "R" is on the net, a new net refiring sequence is started. Depending on the guard, already used transitions can't fire. The deterministic arcs, blocking the guard, keep from further firing sequences for the previous net portion and compell the net behaviour in the same way the cut compells a program computation. The arithmetical functions, which are side-effects to the

Prolog predicates, are side-effects to the transition firings on the nets (see par.3).

6. PROLOG AND THE STRUCTURED-PROGRAMMING LANGUAGES

The relations between Prolog and the structured-programming languages, as C or Pascal, do not die out with a comparison of performance and efficiency. Firstly, Prolog is not a purely logic language, but uses non-logical primitives to perform mathematical and control functions. Even theoretically, a 'puré Prolog doesn't exist.
But this is not a limitation or a shotcoming: it shows instead a possibility of interfacing easily and uniformly other programming languages, which increases the efficiency of the resources management, maintaining the clear Prolog semantics.

Nowadays, the most-diffused Prolog dialects allow the use, through predicates names, of code written mainly in C, which is the side-effect of those predicates.
Semantically, the procedural interpretation becomes central. In fact, this description looks at a generical clause:

$$A(X_1,....,X_n):\text{-}B1\ (Y_{1,1},....{}_{m,1}),..., Br(Y_{r,1},...,Y_{r,mr})$$

as a procedure definition with parameters x1,...xn, computed by calling the subprocedures B1,...,Br with the corresponding parameters. But possibly these procedures can be evaluated not as Prolog procedures: the lower hierarchical subprocedures could be computed with other languages, transmitting only useful information to the upper level.

Prolog hence can demand the computation of deterministic functions to more efficient languages. For these languages instead, Prolog can reflect a nondeterministic structure. Thus, i.e., different procedure definitions can correspond to the same procedure call, so that exceptional situations (for example some situations of error recovery) can be treated. On the BP-Nets, the associations can be used to establish clearly these relations between Prolog and other programming languages, maintaining a homogeneous structure.

The basic nondeterministic nature of Prolog is linked to the problem of the concurrent programming. Prolog is a sequential simulation of a concurrent computation model. The Prolog program is in fact a clause set scanned, during a deduction, with a depth-first sequential search enforced by the resources limitations on the current computers. With infinite resources, naturally the computation would work in a parallel way, activating first all the clauses with a head which unifies the current procedure call and, after that, activating all the calls contained in the procedures body. This pure parallelism, however, could have problems not only with resources, but also, for example, in determining the program termination.
Along this path, the concurrent logic languages are born to use correctly the Prolog features in a concurrent environment.

To use nets becomes now natural because concepts in the field of parallelism and concurrency are clearly shown by Petri nets (causality, concurrency, conflict and confusion), and produce on then a net's hierarchy. Besides a well-known net model, reducible to a Petri nets class, the Kahn-MacQueen model contributes in the definition of the concurrent logic languages.
It is hoped that BP-Nets could be a conjuction between Prolog and nets, enforcing this relation.

7. LOCAL AND GLOBAL INFORMATION IN PROLOG

We would like now to clarify what information is needed by the interpreter to develop the computation and where this information is located with respect to the current evaluation environment. The subgoals evaluations nesting in fact induce an evaluation environments nesting.

BP-Nets point out information transfer and information nesting modes in a Prolog program, aiming to minimize global information and to maximize local information. To clarify this point we have to examine real situations. First, during a computation, the current Byrd operation is determined exploiting the result of the straigth previous operation. This computation is realized using directly available, and hence, completely local information. These are the type of the operation just concluded, the rules examined and the rules that can still be examined).

In the same way, on nets the 0-color of the token to be generated is determined using the connections of the place where the token and the 0-color of the current token. The recursive cycle of unification and evaluation of subgoals needs two choices:

1- to choose a rule between the rules for an objective;
2- to choose a sub-goal to evaluate between the ones of the chosen rule.

As already stated, the order for the rules in 1 - and for the subgoals in 2 - is in PROLOG sequential and 'depth-first'. Now, in situation 1-, the pattern-matching checks if the chosen rule satisfies the actual evaluation constraints.
The values of the arguments of the goal are the needed information during this check, because the rule head arguments are fixed; this is directly available, hence local, information. When the examined rule is activated, the unification realizes an information transfer from the goal to the rule body. This information is exploited in situation 2-.
First, it is needed to know which Byrd operation must be worked out. The operations which enforce an evaluation are CALL and REDO. For CALL, the values of the subgoal arguments come not only from the unifications globally comunicated when the rule is activated, but from the effected unifications in the previous evaluations of the rule sub-goals too, which become globally available to the following sub-goals. REDO, instead, asks for undoing the unifications realized in the immediately previous evaluation concluded with positive exit. In this situation it is needed to know for each argument, if, where and how bindings were realized.

Obviously, this information is global: i.e. it is defined in relation to the comprehensive evaluation process. Infact a binding could be realized in an evaluation environment very far from the current environment. On the BP-Nets the distinction between local and global information passes through the distinction between token and net environment.
The environments are generated or transformed by the transition firings, while the choice of the fired transition depends only on the token value. Just an exception was maintained to simplify the description: the guard constraint for 'R' operation on name-places.
The environment, global to a subnet, is born and dies with its subnet, the stack structure allowethe storage of successively imposed bindings during the net firing sequence. The token, which is the medium for the whole needed information to continue the firing sequence, is directly transferred where is required.

8. PROLOG NET

BP-Nets, were being developed parallely theoretically and through simulation. The simulation was realized by a software tool (called PrologNet), were BP-Nets could be built and the functioning mechanisms of the BP-Nets could be tested.
The two sides of the research supported one another.

PrologNet is developed under Macintosh, exploiting LPA MacProlog 2.0. The choice of this implementation is motivated by the possibility of using a rich set of primitives for graphics and windows management. MacProlog is assumed as reference to define the Prolog programs set which PrologNet can transform into BP-Nets. Currently, the fundamental limitations, according to par.2., are the impossibility of using:

1. functional operators in the predicate parameters;
2. some built-in predicates.

Obviously the predicates which implement graphical and window managing primitives are excluded, because they are not in the core of the definition of Prolog.

Control predicates, input/output predicates, predicates which dynamically transform the program are excluded too. In this development phase, it was preferred do not introduce mechanisms which would transform the firing rule and the net, demanding a detailed examination.
It is possible to use predicates which transform the parameters:

1. metalogical primitives;

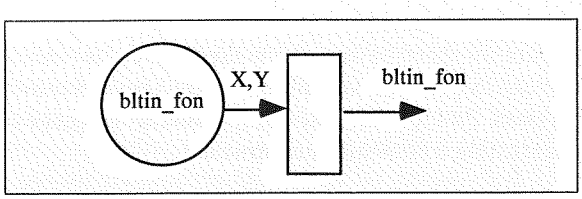

Fig. 8.1

2. type primitives;
3. arithmetical primitives;
4. primitives working on terms and strings.

All these primitives can be theorically seen as association for nets as:
where the transition has an association which is the primitive. During runtime, the operations of expansion and contraction for these subnets are not shown, because they are irrilevant. Hence, in PrologNet the queue-places named with these predicates trigger at once the association of their subnet and give the resulting token. On the screen, PrologNet appears as a menu added to those defined by the MacProlog environment.

There are three commands defined in this menu:

1- 'Draught Machine';
2- 'Enter Interpreter';
3- 'Exit Interpreter'.

'Draught Machine' is used to transform a Prolog object program into in the clauses' set which describe the BP-Nets set equivalent to the program. The command triggers a dialog-box which asks the loading file name and the arity of the main predicate. It presumed that the file name and main predicate name are the same and that every predicate has a fixed arity.
The result of 'Draught Machine' can consist in a nets set, because is generates not only the net equivalent to the main predicate, but the subnets equivalent to the subpredicates of the main predicate too.
The obtained program is saved in a file, transparent to the user, whose name is given from the starting file name + 'dm'.
This operation is automatically used to save memory during runtime. The constraint of available memory for the evaluation space was in fact very severe in the environment where this application was developed. 'Enter Interpreter' asks the name of the net to be simulated. This net must be precedently obtained by 'Draught Machine'. The file corresponding to the net is loaded and every net it contains is written to a graphics window, where the net is drawn.

Graphically, the transitions are ordered from top to bottom on the screen according to the function order, with black deterministic transitions and white nondeterministic transitions. The places are distinguished in this way:

1. name-places are completely white;
2. the queue-places associated to built-in predicates (directly evaluated) have a dark shadowing;
3. other queue-places have a light shadowing.

Using a supplementary window is chosen the net to fire. It is not compulsory to start from the main net.
After choosing a net, the P-color of the initial token is built through dialog-boxes. From here on the net firing sequence proceeds every time the user 'clicks' on the tokens generated by the program on the net. Every click corresponds to an ask to apply the nets transition rule.

The expansion from a queue-place to its subnet is executed carrying in close-up the subnet's window; in the same way a contraction takes the calling net back in close-up. With one command 'Enter Interpreter', more firing sequences for the same nets set can be asked. It is not possible to overlap nets obtained in different sessions of 'Draught Machine'. A file use is concluded by the command 'Exit Interpreter'. 'Exit Interpreter' saves the used file and allows the loading of new files to be used with 'Enter Interpreter' or transformed by 'Draught Machine'.

9. CONCLUSIONS

Following the comparison carried out in [2], some attempts were carried out to deepen the links between logics (and logic programming) and Petri nets. Along this path, some obstacles progressively cropped arise.

First. the Petri nets were defined in a algebraic-mathematical environment, paying a special care to the precision of the formal definitions.
This approach allows the computation of interesting properties and functions, but it is problematic in the quick increase of description complexity for subclasses of nets defined in very specifical environments. For this purpose more powerful models, usually High-Level Petri Nets, are defined to semplify the descriptions. For these models a morfism property holds with lower level nets classes.

In our case, the description of the Prolog computation model can't leave out of consideration control features,

inherited by its nature of programming language. That is why growing the preciseness of the Prolog description, the nets compexity and hence the difficulty of including the obtained class between the Petri net grows. It was preferred to define a very high level nets model (which allows a satisfying description of Prolog), postponing the problem to proof a morfism between these nets and a class of Petri nets.
A draft on the formal definitions for the BP-Nets and on a proof of morfism are yet to be developed.

10. ACKNOWLEDGEMENTS

I would like to thank Prof. Giovanni Degli Antoni for his encouragement and constant support.
I owe Clementz Portmann for greatly improving the English translation, realized in May 1989 at Toyohashi University of Technology. Thanks to Bull Italy: this paper was conceived with its cooperation.

11. BIBLIOGRAPHY

[1] K.L. Clark, F.G. McCabe, N. Johns, C. Spenser, LPA MAC PROLOG REFERENCE MANUAL, Logic Programming Associates Ltd., London, 1988.

[2] H.J. Genrich, PREDICATE/TRANSITION NETS, in (W. Brauer, W. Reisig, G. Rozenberg eds.) Advances in Petri Nets 1986, Vol. I, Lecture Notes in Computer Science 254, Springer Verlag, 1987.

[3] H.J. Genrich, K. Lautenbach, P.S. Thiagarajan, ELEMENTS OF GENERAL NETS THEORY, in (W. Brauer ed.) Net Theory and Applications, Lecture Notes in Computer Scienze, 202, Springer Verlag, 1980.

[4] K. Jense, COLOURED PETRI NETS, in (W. Brauer, W. Reisig, G. Rozenberg eds) Advances in Petri Nets 1986, Vol. I, Lecture Notes in Computer Science, 254, Springer Verlag, 1987.

RAPPRESENTAZIONE COGNITIVA DI STRUTTURE TEMPORIZZATE

Antonio Camurri, Agostino Poggi
DIST - Dipartimento di Informatica Sistemistica e Telematica
Università di Genova

ABSTRACT

This paper presents a model, called Action Nets, for the representation of actions in real world contexts. The model integrates artificial intelligence methodologies, including multiple inheritance frame networks, with analogic simulation methodologies. Action Nets allow the composition of different action descriptions and representations at different levels of detail since an action is represented by a tree of frames which degenerates in a single frame at the lowest level. Besides, this model introduces a formal representation of time and concurrency taken from Petri nets theory; in particular, Timed Colored Petri nets support the simulation of Action Nets.

RIASSUNTO

Questo articolo presenta un modello, denominato Action Nets, per la rappresentazione di azioni nel mondo reale. Il modello integra metodologie di intelligenza artificiale, in particolare reti semantiche ad eredità multipla, con metodologie basate sulla simulazione. Le Action Nets permettono la composizione di differenti modelli di azioni e rappresentazioni a diversi livelli di dettaglio: una azione è rappresentata da un albero di frame che degenera in un unico frame al livello di rappresentazione più basso. Inoltre, questo modello introduce una rappresentazione formale del tempo e della concorrenza che deriva dalla teoria delle reti di Petri; in particolare, reti di Petri colorate e temporizzate supportano la simulazione di azioni descritte mediante Action Nets.

1. INTRODUZIONE

La teoria delle azioni è un settore ampiamente investigato [14], [17]. Approcci classici riguardano la rappresentazione del mondo, di piani e di strategie e si basano su formalismi consolidati che, in particolare, includono la rappresentazione del tempo [10], [5]. Grosse difficoltà in tali approcci si incontrano nel trattamento di *azioni nel tempo* in contesti reali complessi: ad esempio, in robotica, in applicazioni "business-oriented", nella musica, in problemi di pianificazione, ci si scontra spesso con ambienti o contesti parzialmente sconosciuti o mutevoli, obiettivi non completamente specificati, pianificazioni non completamente deterministiche.

Il problema di gestire descrizioni incomplete è ben noto [18]: tuttavia, pur esistendo metodologie in grado, ad esempio, di generare pianificazioni molto complesse, spesso si dimostrano scarsamente capaci di gestire anche semplici azioni che coinvolgono monitoraggio e retroazione interattiva sull'ambiente, cosa molto comune in problematiche reali.

Un ulteriore punto debole degli approcci tradizionali riguarda la capacità di rappresentare *skill* [1], ovvero "abilita" generali acquisite e memorizzate relative a come risolvere una data classe di problemi. Uno *skill* è la controparte attiva del *common sense* [12]: la capacità di risolvere dei compiti applicando una strategia generale e riutilizzabile in ogni problema (non necessariamente definito completamente).

Il presente lavoro si prefigge di integrare alcuni aspetti derivati dalle precedenti metodologie in un approccio euristico al problema della rappresentazione di azioni:

in particolare, vengono presentati i presupposti metodologici ed alcuni risultati di un modello ibrido, denominato *Action-Nets* (A-Nets) [8], che abbiamo recentemente applicato nei settori della robotica [2] e della rappresentazione della conoscenza musicale [9]. Di questo modello é stato sviluppato la parte relativa al ragionamento, con tecniche sia di reti semantiche che di programmazione ad oggetti, integrate con una formale *rappresentazione del tempo* e della concorrenza, la cui semantica é derivata dalla teoria delle reti di Petri (Petri Nets - PN) [21].
Il paragrafo successivo introduce la struttura generale del modello con la definizione del modello di PN utilizzato (*High Level Timed PN*) e lo illustra attraverso un esempio nel settore della robotica, riguardante le azioni di un robot in ambienti mutevoli, non strutturati e con rappresentazioni non esaustive di problemi; il terzo paragrafo discute il rapporto esistente tra i linguaggi ad oggetti e il modello A-Net; il quarto paragrafo descrive il modello in termini di base di conoscenza e semantica della rappresentazione.
Nel quinto paragrafo viene analizzato in dettaglio il meccanismo per passare dalla rappresentazione di azioni in termini di reti semantiche ad una in termini di PN. Segue nel sesto paragrafo una breve discussione sulle possibilità di ragionamento basato su modelli ad azioni sfruttando la teoria delle PN. Alcune note sulle linee di ricerca attualmente perseguite concludono l'articolo.

2. RAPPRESENTAZIONE DI AZIONI NEL TEMPO

Una azione complessa può essere modellata in termini di eventi discreti, che rappresentano cambiamenti asincroni e *causali* nella struttura di un particolare contesto. Se per comodità di esemplificazione, si fa riferimento alle applicazioni robotiche, esempi in un' azione di un robot potrebbero essere: i) una collisione; ii) un secondo robot, o parte del robot stesso, si trova in una posizione rilevante; iii) un oggetto entra nel campo visivo; iv) una forza od una posizione ha raggiunto un particolare valore. In tutti i casi la causalità può essere rappresentata dalle regole specificanti il modo in cui processi continui sono "connessi" da eventi. All'interno dei processi, la *causalità* può essere gestita in altri modi, ad esempio mediante simulazione quantizzata (quantitativa o qualitativa).
Il modello di azioni nel mondo reale è tipicamente a quattro dimensioni (spazio+tempo). Un *evento* è un "punto speciale" sull'asse temporale, che connette due "sotto-modelli"; ciò è analogo a quanto accade in un modello tridimensionale, dove un oggetto può essere pensato come due sotto-oggetti correlati geometricamente. Tuttavia, nel mondo reale, nessun modello tridimensionale può essere decomposto in sotto-modelli *totalmente indipendenti*; alla stessa maniera, due *processi* connessi attraverso *eventi* sono solo un' approssimazione della realtà. Da un altro punto di vista, l'intero sistema può essere considerato come un insieme di rappresentazioni parziali (eventualmente continue), di cui solo un sottoinsieme è simultaneamente attivo. Quindi, una rappresentazione formale di un'azione richiede non solo la descrizione dei suoi "processi", ma anche dell'insieme degli "eventi" rilevanti per la sua evoluzione, relativamente allo stato del mondo e/o ad altre azioni cooperanti.

In generale, modelliamo un' *azione* come una cooperazione di entità concorrenti e asincrone che indicheremo nel seguito con il termine *processi*.
Un processo produce cambiamenti nel mondo (attraverso movimenti, posizionamenti, ecc.) o rilevamenti (attraverso attività sensoriali e di riconoscimento a basso livello): in quest'ultimo caso, il processo viene chiamati *percettore*. Ogni processo ha un proprio stato interno e può accedere ad uno *stato globale* ed al modello dell'ambiente o contesto.

Il ruolo dei processi, in applicazioni robotiche, è quello di modellare la cinematica, la dinamica, i processi fisici, la percezione ecc., integrando rappresentazioni *procedurali e dichiarative*. Possiamo anche pensare a dei processi a più alto livello (*ragionatori*) in grado di compiere ragionamenti, pianificazioni, riconoscimento ad alto livello e così via. Un modello del genere richiede qualche definizione sulle interazioni tra processi e sulla loro semantica.

2.1 Reti di Petri ed Azioni

Le azioni sono reti di processi, le cui proprietà sono essenzialmente quelle delle reti di Petri temporizzate [4] ad alto livello [16]. Il formalismo delle reti di Petri costituisce la base dello "scheletro temporale" delle azioni. L'uso delle reti di Petri come struttura basilare per modellare azioni nel tempo è dovuto sia alla potenza

descrittiva del formalismo che alla grande mole di risultati già ottenuti con esse: i teoremi su connettività e raggiungibilità, ad esempio, possono fornire un potente strumento per ragionare su un modello ad azioni e ricavarne proprietà.
Inoltre, la teoria delle reti di Petri consente di rappresentare sia *processi* - caratterizzati da intervalli temporali in cui essi avvengono - che *eventi* - caratterizzati da particolari istanti significativi.
Il modello A-Nets da noi sviluppato è basato su *High Level Timed Petri Nets* a più livelli di astrazione (i posti possono rappresentare sotto-reti di Petri), con durate temporali associate ai posti/processi. Nel seguito assumeremo come nota la terminologia relativa alle reti di Petri.
Una rete di Petri costruita partendo da un modello A-Nets è caratterizzata da tre posti predefiniti, per identificarne l'inizio (Start_place), una eventuale terminazione corretta (End_place) od un suo fallimento (Failure_place). Una esecuzione a buon fine termina sempre con una marca (od una per ogni colore nella rete) su End_place. E' da notare che una sotto-azione (rappresentata con un posto ad alto livello) la cui esecuzione fallisce causa il fallimento dell'intera azione (vedi il meccanismo a sottoreti nell'algoritmo descritto nel paragrafo 4.1).

Nel nostro modello, i posti rappresentano: (i) condizioni (come in Condition Event systems, caso particolare di reti di Petri [21]), predicati o contatori con una data capacità; oppure, (ii) istanze di processi, rappresentati da *frame*.

Nel primo caso, un posto non ha durata temporale, e rappresenta tipicamente un sensore (reale o virtuale), il cui stato (marcatura) è dato dalla valutazione di espressioni logiche operanti su dati esterni rappresentati, a livello di A-Nets, mediante attributi (proprietà) di particolari oggetti/frame (per esempio, misure nella rappresentazione di un ambiente tridimensionale in cui sta operando un robot).

I posti che rappresentano processi sono caratterizzati da un intervallo temporale, corrispondente alla durata di esecuzione del processo.

Le transizioni gestiscono le sincronizzazioni fra istanze di processi e ad esse non sono associate durate temporali; esse vengono eseguite non appena la regola di scatto è verificata, producendo immediatamente le marche in uscita.

Posti e transizioni sono connessi tramite archi; per migliorare il potere espressivo, sono stati inclusi nel modello anche *archi inibitori*.

Le marche sono coppie di valori

(Color, Time)

dove *Color* identifica un *agente* e *Time* è un valore intero e positivo: un posto è marcato dall'agente A se contiene una marca il cui colore è eguale ad A e *Time* è minore od uguale a *WorldTime* (il valore corrente del tempo nel mondo rappresentato).
Le Colored PN [16] consentono di riunire in una singola PN diverse istanze della stessa classe di azioni eseguite da differenti agenti: ogni agente è rappresentato nella rete dalle marche corrispondenti al suo colore.
Una volta definita una classe di azioni per un agente, è possibile generalizzarla per N agenti che effettuano lo stesso tipo di azione introducendo N colori nella rete, ed introducendo N marche in Start_place, una per ogni colore.

Le transizioni sono caratterizzate dalla seguente regola di scatto: una transizione scatta non appena *tutti* i suoi posti in ingresso risultano marcati e le eventuali marcature dei posti in uscita non eccedono le loro capacità.
Inoltre, una transizione può scattare simultaneamente per più Agenti/Colori se essi hanno marcature di ingresso e di uscita disgiunte; questo corrisponde allo scatto simultaneo della stessa transizione in differenti istanze della stessa azione per differenti agenti.
Non è necessario, a livello di PN, fornire maggiori dettagli sulla regola di scatto per le transizioni: eventuali *conflitti od alternative* (vedi figura 1) sono gestiti ad

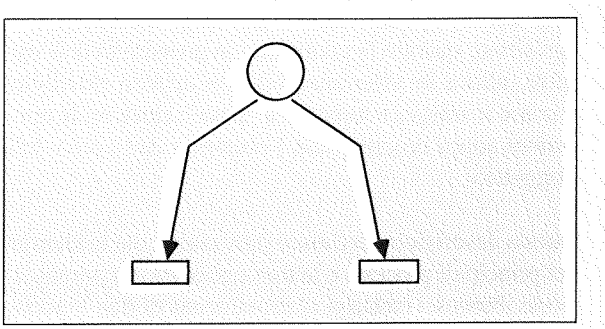

Fig.1 Caso di conflitto od alternativa in reti di Petri.

un livello più alto.

In caso di conflitti, possiamo rinviare la soluzione alla parte del sistema a più alto livello: ogni conflitto in una PN é risolto introducendo sensori "virtuali", che hanno funzione analoga a quella svolta dal posto *Oggetto_Afferrato* in figura 3: senza tale posto, in tale PN si avrebbe un caso di conflitto nel posto *Afferra*; la presenza di *Oggetto_Afferrato* risolve il caso di conflitto. La marcatura di questo nuovo sensore "virtuale" deriverà dalla valutazione di particolari predicati, riguardanti particolari strategie sul modo si eseguire l'azione.

2.2 Un Esempio di Azione

Introduciamo a questo punto un esempio, relativo ad una azione "Afferra e Posa" (per semplicità considerata unicamente in termini cinematici) consistente di due fasi e così definita:

<un uomo afferra un oggetto senza muovere i piedi, mantenendo l'equilibrio ed evitando l'interferenza dei volumi. L'azione deve poter modellare la presa e la posa di un oggetto da ogni posizione a un'altra qualsiasi se cinematicamente possibile>.

Una possibile descrizione qualitativa per l'azione é la seguente:

<stima della distanza tra l'uomo e l'oggetto, seguita dalla decisione su come partizionare il posizionamento mediante il piegamento di *braccia*, *tronco* e *gambe*; concorrentemente viene iniziato il movimento di *braccia*, *tronco* e *gambe*, e vengono eseguite le seguenti operazioni:
i) orientare il movimento delle mani verso l'oggetto in questione; ii) evitare collisioni tra le parti del corpo; iii) muovere il corpo in modo da soddisfare le condizioni di equilibrio; quando le mani sono in prossimità dell'oggetto, allora lo afferrano; iv) non appena possibile, iniziare le misure relative alla "posa"; ripetere, le operazioni sopra descritte con l'obiettivo della posa dell'oggetto>.

Questa descrizione definisce informalmente le identità dei principali processi e le loro necessità di comunicazione. Processi (relativi al movimento di due braccia, due gambe, tronco, due mani) e percettori (quattro istanze di rilevatori di collisioni per le braccia e le gambe, un rivelatore di stato di equilibrio ed uno stimatore/decisore per le mani) possono essere utilizzati per costruire l'azione descritta. A tale scopo occorre definire delle connessioni per lo scambio di messaggi (eventi e loro semantica) tra le entità.

L'azione generale è pertanto composta da: i) un insieme di processi, ii) un insieme di connessioni, iii) la semantica delle connessioni. A sua volta ogni processo può essere espanso in sotto-processi fino al livello di astrazione desiderato. La A-Net relativa all'azione dell'esempio trattato è mostrata in figura 2a: gli oggetti presenti nella figura sono ciascuno il risultato della organizzazione tassonomica di cui è mostrato un esempio semplificato in figura 2b relativamente all'oggetto Muovi_Mano_Dx. In particolare, occorre notare che le connessioni tra oggetti in figura 2a non rappresentano legami di eredità (vedi paragrafo successivo) bensì lo schema ad A-Nets di organizzazione di azioni come definito in seguito (paragrafo 4).

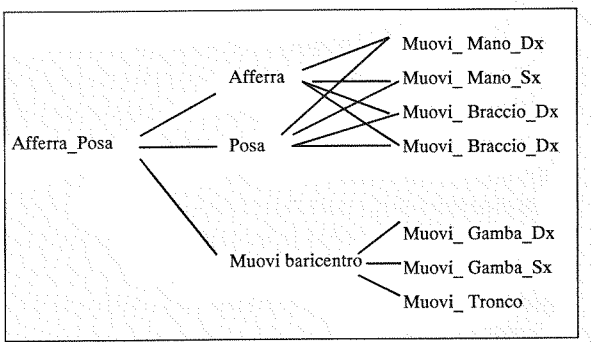

Fig.2a A-Net relativa alla azione "Afferra e Posa".

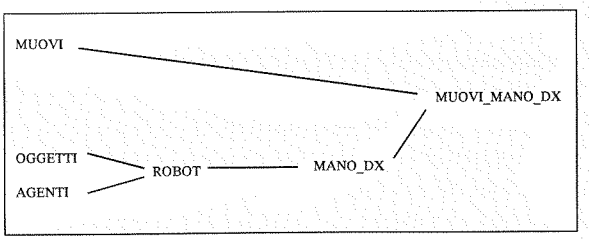

Fig.2b Rete semantica relativa alla costruzione del frame Muovi_Mano_Dx.

Il modello a quattro dimensioni può essere visto anche in relazione alla pianificazione di un'azione: in figura 3, è mostrata la rete di Petri ad alto livello relativa al precedente esempio, in cui ad ogni processo è associa-

to un posto (i posti indicati da cerchi corrispondono a processi, i posti esagonali a sensori). Nel seguito verrà mostrato come da una descrizione di una classe di azioni in termini di A-Nets sia possibile generare *automaticamente* un modello relativo ad una istanza in termini di reti di Petri.

3. LINGUAGGI ORIENTATI AGLI OGGETTI ED A-NETS

Gli oggetti sono entità che combinano le proprietà delle procedure e dei dati (chiamati rispettivamente metodi e variabili). Il comportamento di un oggetto è definito dai suoi metodi, in quanto un oggetto può essere manipolato solo attraverso di essi.
Seguendo una classificazione molto usata, suddividiamo gli oggetti in due categorie: classi e istanze. Una classe è una descrizione di uno o più oggetti simili (una classe corrisponde al concetto di *tipo* in linguaggi di programmazione procedurali), un'istanza è una particolare entità di una classe.
L'eredità è una tecnica che permette di costruire nuove classi in base a quelle già esistenti; la nuova classe viene chiamata sottoclasse, quella (o quelle) da cui eredita le slot[1] viene chiamata superclasse. I linguaggi orientati agli oggetti sono classificabili attraverso il tipo di eredità:

1) eredità gerarchica: una classe è definita in termini di una singola superclasse (Smalltalk [13]);
2) eredità multipla: una classe può ereditare da più di una classe (Flavors [19] e le Unit del Kee [11]).

Il livello di A-Nets è basato su reti semantiche di oggetti con meccanismi di eredità multipla (vedi figura 2b). Gli

[1] Con il termine slot si intendono sia i metodi che le variabili di un oggetto.

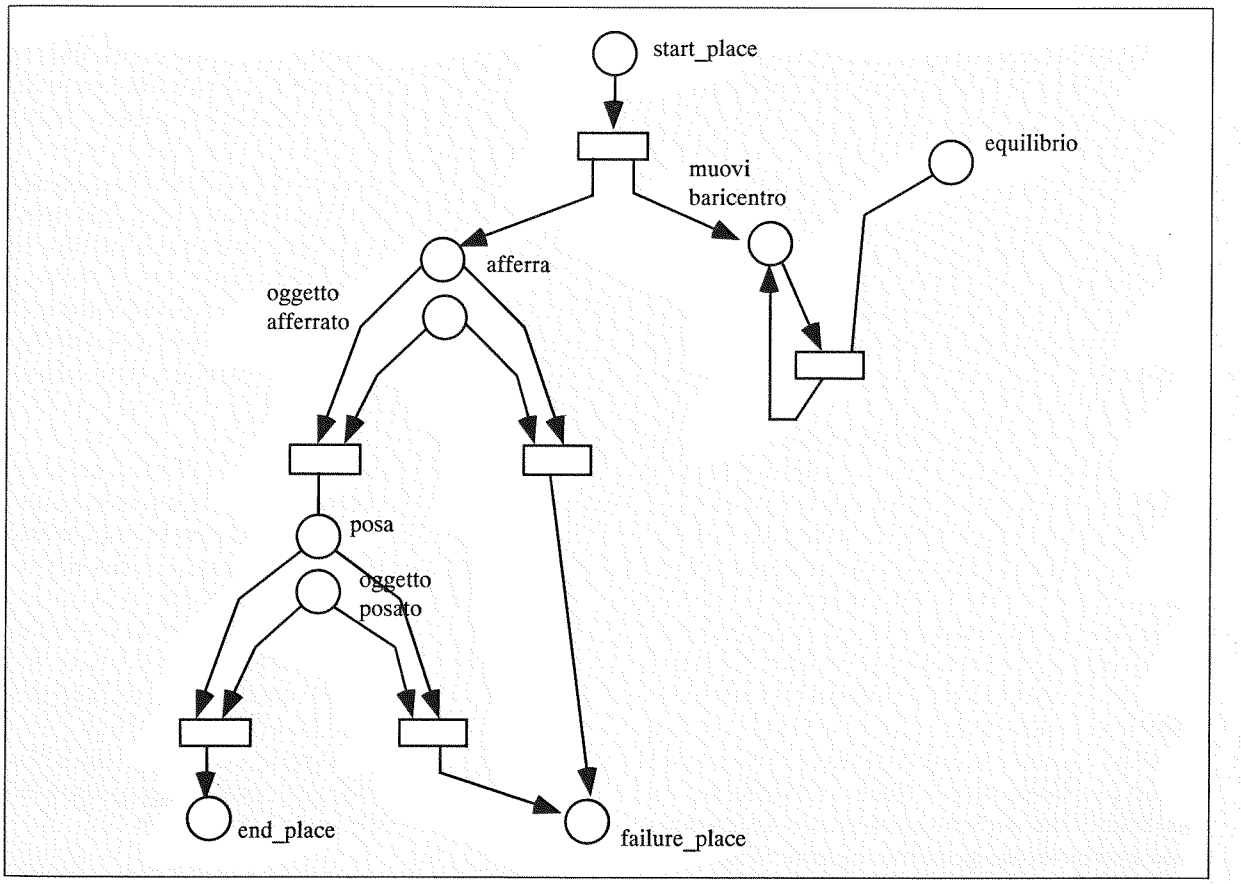

Fig.3 Rete di Petri della azione "Afferra e Posa".

oggetti nel modello A-Nets sono usati per due ragioni; la prima è che l'eredità permette di costruire facilmente oggetti/azioni a diverso livello di dettaglio sicchè una nuova azione a più alto livello di dettaglio è ottenibile mediante poche modifiche incrementali rispetto ad un modello esistente; la seconda ragione è che l'accesso esclusivo ad un oggetto attraverso i suoi metodi permette una buona astrazione dei dati, e quindi i programmi che accedono ad un oggetto non devono conoscerne la rappresentazione interna. Nel nostro caso questa proprietà è utile per costruire nuovi oggetti componendo dei modelli già esistenti conoscendo solamente il loro protocollo esterno.

4. DESCRIZIONE DEL MODELLO A-NETS

Il modello A-Nets consente di rappresentare la conoscenza relativa a quali azioni devono essere connesse e come, allo scopo di costruire una nuova azione ad un piu' alto livello di astrazione.
Una *A-Nets* è la rappresentazione astratta, simbolica di classi di azioni basate su reti di Petri. Quindi, le A-Nets sono ad un livello di astrazione più alto rispetto alle PN [8].

Le A-Nets sono realizzate da un sistema ad oggetti composto di:

1) una *Long Term Memory* (LTM), contenente la conoscenza del mondo in termini di azioni e processi;
2) una *Working Memory* (WM), contenente istanze di azioni.

In questo caso, il termine processo indica sia un' attività atomica rappresentata da un unico oggetto (*primitive*) che un' attività eseguita da più processi (*action*). La esecuzione di un' azione è determinata dalla esecuzione dei vari processi componenti, sincronizzati attraverso messaggi. Un'azione è rappresentata da un albero di oggetti, la cui radice contiene la rappresentazione dell'azione; gli oggetti rimanenti contengono le rappresentazioni dei processi che compongono l'azione. Un processo può partecipare ad azioni differenti: l'oggetto che lo rappresenta quindi risulterà collegato agli oggetti rappresentanti tali azioni. Un processo può comportarsi come una sotto-azione (rappresentato quindi da un sotto-albero di oggetti) permettendo la rappresentazione di una stessa azione a diversi livelli di dettaglio.

Nell'esempio iniziale (figura 2a), *Afferra*, *Posa* e *Muovi_Baricentro* sono gli oggetti dei processi ad alto livello dell'azione: essi possono essere considerati, a seconda del livello di dettaglio, processi primitivi o sottoazioni composte a loro volta dai processi *Muovi_Braccio_Dx*, *Muovi_Braccio_Sx* *Muovi_Mano_Dx*, *ecc.*.

I processi sono connessi da *fili* (*wires*), che trasmettono messaggi dotati di associata una propria semantica; questi messaggi hanno due scopi: i) sincronizzare i processi (un processo attiva un altro processo attraverso l'invio di un messaggio); ii) scambiare dati.
Esiste una classe "padre" di ogni altro oggetto contenente le proprietà generali per qualsiasi tipo di oggetto: la classe *ACTIONS*. Essa contiene le seguenti *slot* (proprietà generali ereditate dalle azioni):

BEHAVIOURS (lista di valori di tipo primitive / action) un processo (oggetto) si comporta, per ogni agente, come un oggetto primitivo oppure come una sotto-azione a seconda della circostanza e del livello di dettaglio;

AGENTS (usato se BEHAVIOUR uguale ad action) contiene la lista degli agenti che possono compiere l'azione: essi corrisponderanno, a livello di PN, al l'insieme dei *colori* possibili per le marche. Ogni agente è rappresentato dal flusso di marche relative al proprio colore.

PROCESSES (usato se BEHAVIOUR uguale ad action) contiene la lista dei processi che compongono la sotto-azione (essi corrispondono a posti/processi a livello di rete di Petri);

SENSORS (usato se BEHAVIOUR uguale ad action) contiene la lista dei posti/predicati (livello di rete di Petri), corrispondenti ai sensori coinvolti nell'azione per il monitoraggio di possibili eventi e/o dati esterni;

WIRES (usato se BEHAVIOUR uguale ad

action) contiene la lista delle connessioni dell'azione;

PN_CODE (usato se BEHAVIOUR uguale ad action) slot in cui viene memorizzato il codice della rete di Petri generato dall'algoritmo di creazione (descritto nel seguito);

DATA lista dei parametri dell'azione;

PRIMITIVE_CODE (usato se BEHAVIOUR uguale ad primitive) parte procedurale associata all'processo primitivo;

TIME (usato se BEHAVIOUR uguale ad primitive) durata temporale di PRIMITIVE_CODE: se essa non è conosciuta a priori, viene calcolata durante l'esecuzione di PRIMITIVE_CODE.

A seconda del valore corrente di BEHAVIOUR, viene usato solo un sottoinsieme di slot di un oggetto: per esempio, PRIMITIVE_CODE e PN_CODE sono mutuamente esclusive; quindi un oggetto può contenere sia la descrizione di una azione primitiva (*primitive*) che la descrizione di una azione composta (*action*).

Ogni volta che creiamo una istanza di una azione (BEHAVIOUR uguale ad *action*), la slot PN_CODE verrà inizializzata al codice relativo alla generazione della rete di Petri corrispondente alla istanza voluta della azione.
Occorre notare che ogni oggetto della base di conoscenza (sia esso processo od azione) deve essere connesso - direttamente od indirettamente - alla classe ACTIONS, per ereditare le proprietà comuni sopra definite.
Proseguiamo ora l'esempio iniziale, entrando più nei dettagli della azione Afferra_e_Posa[2]:

Azione **Afferra_e_Posa**:

BEHAVIOUR	Action
AGENTS	(robot1)
PROCESSES	(Afferra Posa Muovi_Baricentro)
SENSORS	(Equilibrio Oggetto_Afferrato Oggetto_Posato)
WIRES	(((is-marked Start_place) (t (link'Afferra) (link' Muovi_Baricentro))) ((all-marked Afferra Oggetto_Afferrato) (t (link 'Posa))) ((all-marked Afferra (not Oggetto_Afferrato)) (t (link 'Failure_place))) ((all-marked Muovi_Baricentro (not Equilibrio)) (t (link 'Muovi_Baricentro))) ((all-marked Posa (not Oggetto_Posato)) (t (link 'Failure_place))) ((all-marked Posa Oggetto_Posato) (t (exit)))
PN_CODE	nil
DATA	lista dei parametri

L'azione è eseguita in una unica istanza da un unico agente, rappresentato da un flusso di marche di colore *robot1*.

La parola chiave *link* indica la connessione di un processo o di una sottoazione; la parola chiave *exit* indica l'uscita dalla rete o dalla sottorete (nel caso di una sotto-azione) di Petri corrispondente all'azione.

Le slot WIRES è una lista i cui elementi hanno la seguente struttura: (<firing rule> (< condition > < messages >) ... (< *condition* > < *messages* >)). La <firing rule> (associata alla transizione a livello di rete di Petri) implementa la regola di attivazione della transizione come descritto in precedenza. Una volta scattata, una transizione attiva i messaggi la cui <condition> è vera.
Un esempio della struttura di un processo *primitivo* è dato dall'oggetto di *Muovi_Braccio_Dx*:

Azione Muovi_Braccio_Dx:

BEHAVIOUR	Primitive
DATA	parametri ereditati da Afferra e Posa
PRIMITIVE_CODE	codice per la esecuzione di Muovi_Braccio_Dx
TIME	durata dell'processo

Un secondo esempio riguarda il moto di una palla che rimbalza su un piano: la relativa A-Nets, cui diamo il

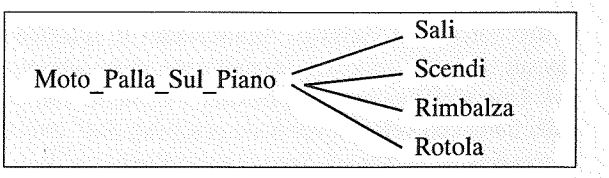

Fig. 4 A-Net della azione "Moto_Palla_Sul_Piano".

[2] In questo esempio e nel seguito dell'articolo verrà utilizzato un linguaggio à la Lisp.

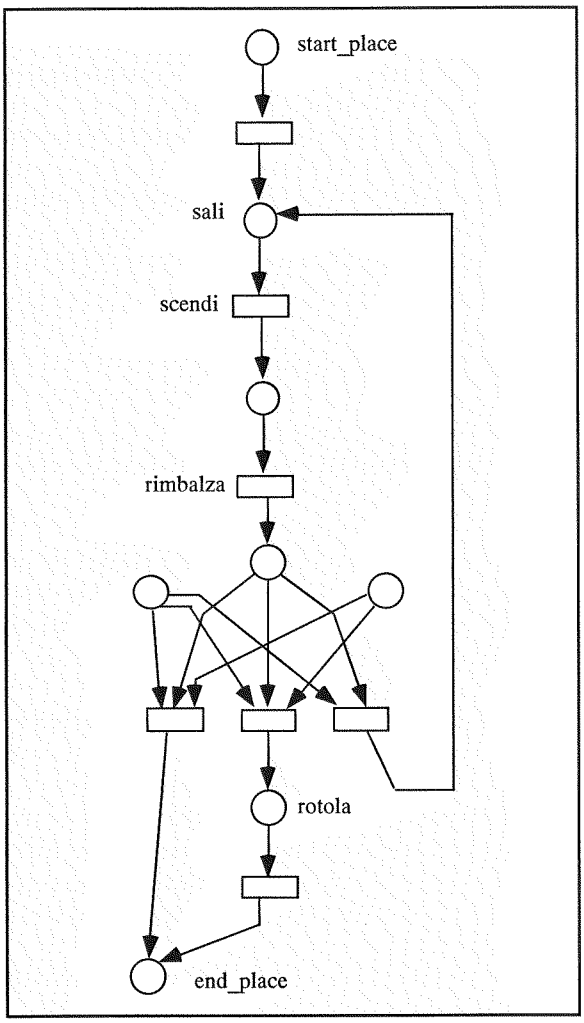

Fig.5 Rete di Petri della azione "Moto_Palla_Sul_Piano".

nome di *Moto_Palla_Sul_Piano*, è mostrata in figura 4,

mentre una possibile rete di Petri dell'azione è mostrata in figura 5.

L'azione *Moto_Palla_Sul_Piano* è composta da quattro processi *Sali*, *Scendi*, *Rimbalza* e *Rotola*. La sua esecuzione agisce su un modello della palla che nel nostro caso è un oggetto. Analizziamo le relazioni causali tra i vari processi: inizialmente viene attivato *Sali* (supponiamo per semplificare l'esempio che la palla parta con velocità verticale positiva), *Sali* attiva successivamente *Scendi*, *Scendi* attiva *Rimbalza*; *Rimbalza* attiva *Sali* se la velocità verticale è maggiore di zero, attiva *Rotola* se l'unica componente della velocità diversa da zero è quella orizzontale, fa terminare l'azione se la velocità è nulla; *Rotola* termina sempre l'azione. Di seguito è mostrato il contenuto degli oggetti *Moto_Palla_Sul_Piano e Sali.*

Azione **Moto_Palla_Sul_Piano**:

BEHAVIOUR	Action
PROCESSES	(Sali Scendi Rimbalza Rotola)
SENSORS	(Null_Speed_X Null_Speed_Y)
WIRES	(((is-marked Start_place) (t (link Sali))) ((is-marked Sali) (t (link 'Scendi))) ((is-marked Scendi) (t (link 'Rimbalza))) ((all-marked Rimbalza (not Null_Speed_Y)) (t (link 'Sali))) ((all-marked Rimbalza Null_Speed_Y (not Null_Speed_X)) (t (link 'Rotola))) ((all-marked Rimbalza Null_Speed_Y Null_Speed_X) (t (exit))) ((is-marked Rotola) (t (exit)))
DATA	lista dei parametri

Azione **Sali**:

BEHAVIOUR	Primitive
DATA	parametri ereditati da Moto_Palla_Sul_Piano
PRIMITIVE_CODE	codice per la esecuzione di Sali
TIME	durata del processo

5. GENERAZIONE DI ISTANZE DI AZIONI

Il modello ad A-Nets consente di ragionare su proprietà di azioni, con riferimento sia a relazioni causali e temporali (mediante la rappresentazione in termini di reti di Petri) che a relazioni di tipo logico-simbolico; inoltre consente di effettuare misure sul modello, basandosi, tra l'altro, su capacità simulative di istanze di azioni.
In questo paragrafo ci occuperemo di definire un meccanismo per passare dal livello A-Nets di un'azione al livello PN relativo ad una istanza dell'azione; inoltre verranno descritte le capacità simulative del modello, rimandando al paragrafo successivo una discussione sul reasoning.

Per eseguire una azione occorre: i) creare una istanza dell'azione nella Working Memory; ii) creare il codice eseguibile della rete di Petri corrispondente, in base alle connessioni dell'azione; iii) eseguire questo codice. Le applicazioni da noi attuate hanno fatto ricorso al linguaggio Lisp.

Il codice di una azione può essere costruito facilmente dai suoi processi e messaggi in termini di PN; i processi corrispondono ai posti, le transizioni sono gli elementi di sincronizzazione tra i processi (attivazione dei messaggi).
Come accennato in precedenza, in una PN sono presenti tre posti che non si riferiscono a processi: *Start_place, End_place e Failure_place.*
Una delle operazioni fondamentali da compiere è il passaggio da una rappresentazione in termini di reti semantiche ad una in termini di PN: a tale scopo occorre generare, in modo automatico, del codice che rappresenti sia la topologia della azione/PN che le modalità di esecuzione.
Un possibile algoritmo adatto a questo compito può essere sintetizzato nel modo seguente:

1. Generazione del codice di inizializzazione della PN:
   1.1 world_time = 0
   1.2 places_list ={Start_place End_place Failure_place}
   1.3 Start_place = MARKED
   Start_place contiene una marca colorata per ogni agente definito nella lista AGENTS.
   1.4 End_place = NOT_MARKED
   1.5 Failure_place = NOT_MARKED

2. Generazione dei posti e delle relative marcature iniziali:
   2.1 *Add* {ACTORS SENSORS[3]} *to* places_list
   2.2 *For Each* place in {ACTORS SENSORS} *do*
   place = NOT_MARKED

3. Generazione della topologia (wiring) e del codice relativo alla PN:
   3.1 generazione del codice relativo ai sensori:
   For Each element in SENSORS *do*
   generazione del codice relativo a condizioni esterne (monitoring)
   3.2 generazione del codice relativo alle transizioni:
   For Each element in WIRES *do*:
   For Each agent in AGENTS *do*:
   3.2.1 definizione della firing rule,
   3.2.2 codice per il consumo delle marche nei posti di ingresso,
   3.2.3 codice per la produzione delle marche nei posti di uscita.

4. Espansione ricorsiva dei posti ad alto livello (sotto-azioni) nelle corrispondenti sottoreti di Petri:
   for each place in places_list *do*
   if BEHAVIOUR of place is equal to *action*, *then*
   place viene espanso ricorsivamente nella corrispondente sottorete (esecuzione ricorsiva dei passi 2. e 3.).
   Connessione della sottorete nella rete originaria[4]:
   4.1 Start_place e la relativa transizione in uscita vengono rimossi; gli archi in uscita da tale transizione sono connessi come archi in uscita ad ogni transizione di ingresso a place.
   4.2 Failure_place viene a coincidere con il posto Failure_place della rete originaria, quindi tutti gli archi connessi a tale posto nella sottorete sono spostati sul Failure_place della rete originaria.
   4.3 End_place cambia il nome in End_place_<subnet name> ed è connesso alle trasizioni di uscita di place.

5. Risoluzione di Conflitti:
   5.1 Identificazione delle transizioni in conflitto
   5.2 Introduzione di Sensori Virtuali come posti di ingresso a tali transizioni.

6. Generazione del codice di terminazione:
   6.1 Aggiornamento tempo corrente:
   world_time = world_time + 1
   6.2 *if* End_place or Failure_place MARKED *then*
   Gestione della terminazione della azione

A livello di rete di Petri (istanze di azioni), un posto può corrispondere ad un processo oppure ad un sensore: in quest'ultimo caso, il posto conterrà dei predicati riguardanti eventi e/o dati esterni alla azione corrente, il cui valore influirà sulla marcatura del posto stesso.
L'esecuzione dell'azione è data dal flusso di marche nella rete. Lo scatto di una transizione causa l'attivazione del codice (PRIMITIVE_CODE) associato ai processi/posti in uscita dalla transizione; ogni processo genera nel posto associato una marca di valore uguale al tempo corrente incrementato della durata del codice del processo, contenuta nella slot TIME del frame del processo.
L'esecuzione del codice generato dall'algoritmo di creazione, ridotto all'essenziale, ha la seguente struttura:

[3]Più correttamente, la lista dei sensori ricavata dalla lista SENSORS, contenente anche la parte procedurale associata ad ogni sensore.

[4]Start_place, End_place, Failure_place sono riferiti alla sottorete, place è il posto da espandere nella rete originaria

(1) marcatura iniziale della rete;
(2) Finchè non viene marcato End_place oppure Failure_place:
- schedulazione delle transizioni;
- aggiornamento del ciclo;
(3) fine.

6. REASONING ED A-NETS

La generazione della PN di un'azione consente, oltre alla simulazione, altre possibilità. Per esempio, è possibile, mediante la istanziazione di opportuni ragionatori (anch'essi in forma di PN), porre domande al sistema quali: "Esiste un processo Y attivato simultaneamente al processo X...?" oppure "In quali casi l'azione va incontro a fallimento nelle ipotesi...?". Esempi di PN in grado di effettuare semplici ragionamenti temporali (temporal reasoner) sono mostrati in figura 6. I posti indicati da esagoni rappresentano dei "ganci" mediante i quali connettersi alla particolare istanza di azione[5].

Altri tipi di attività che coinvolgono ragionamento riguardano esplorazioni della azione/PN (sia tramite analisi che simulazione) per verificare la raggiungibilità di un certo obiettivo, il calcolo di invarianti, e comunque ogni tipo di analisi consentita dalla teoria delle PN.
Quest'ultima affermazione necessita di un breve commento riguardo al modello di PN definito in precedenza: i posti di tipo *sensore* hanno un comportamento, rispetto alla marcatura, non standard.
Infatti, in essi possono apparire o scomparire marche durante la simulazione, in funzione di eventi esterni al sistema.
In queste condizioni è ovvio che teoremi e proprietà delle PN non sono applicabili. Il problema può essere risolto:

1) facendo rientrare nel modello anche la rappresentazione degli eventi esterni che influenzano i sensori: occorre quindi modellare il comportamento dei sensori per esempio mediante processi stocastici [20], rientrando nella teoria delle PN Stocastiche.

[5]E' da notare che i due esempi mostrati in figura 6 rappresentano i predicati temporali corrispondenti alle primitive *Equal*, *Before* e *After* della teoria di Allen.

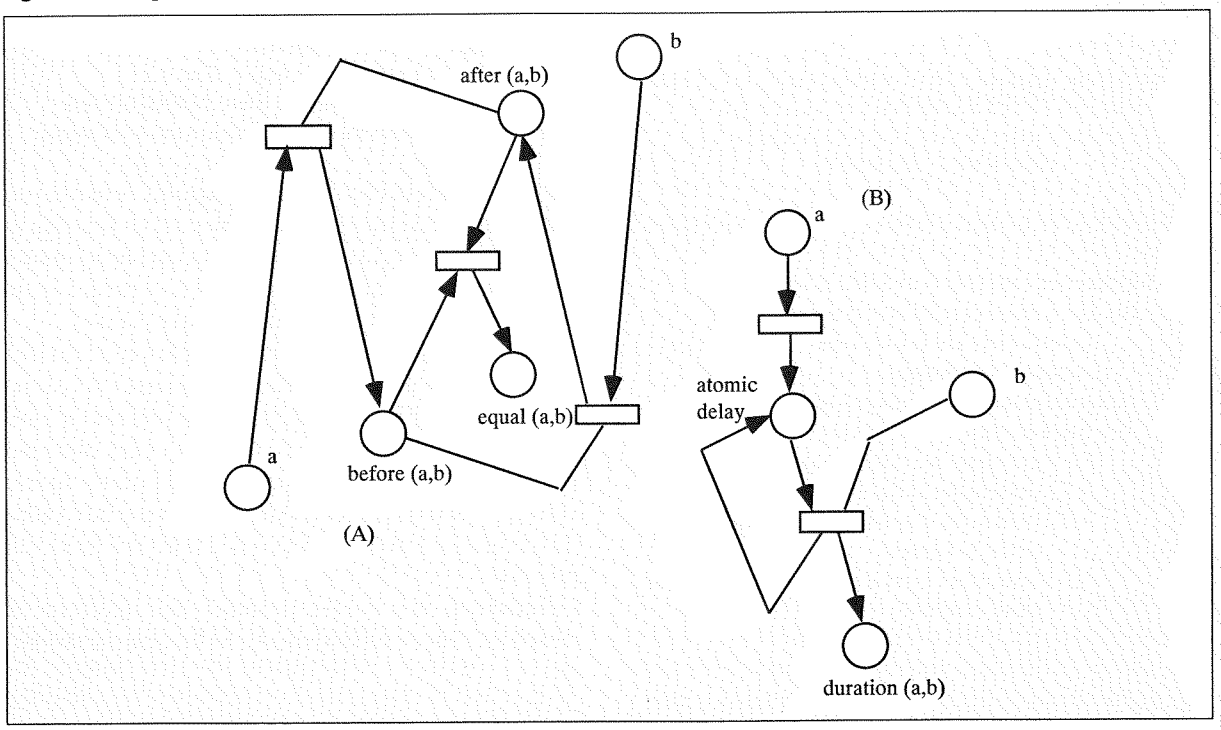

Fig.6 Esempi di Temporal Reasoner.

2) facendo ricorso a rappresentazioni parziali del mondo, basate su ipotesi che si possono fare sul comportamento futuro del mondo stesso. Una descrizione parziale del mondo è una PN in cui sono state fatte delle ipotesi sul comportamento del modello (dei sensori in particolare), disabilitando il meccanismo di auto-modifica della marcatura nei sensori.

Nell'ambito di quest'ultima soluzione è possibile definire, per applicazioni di *planning*, particolari descrizioni parziali del mondo. Ad esempio, seguendo la teoria di Allen [5], potremmo definire le seguenti tre classi di descrizioni parziali su cui effettuare ragionamenti: *expected world, desired world e planned world*, considerate per un singolo agente. L' *expected world* è ciò che l'agente crede accadrà, supponendo l'inattività da parte di ogni agente nel modello (tuttavia le cose che stanno cambiando, i processi attivi, continuano nella loro attività).
Il *desired world* è ciò che l'agente desidera che accada, ad esempio in termini di marcatura della PN.
Il *planned world* è come l'*expected world*, tuttavia l'agente in questione può aggiungere il suo intervento nella PN.
Un possibile obiettivo è che il *planned world* "assomigli" il più possibile al *desired world*.

Risulta pertanto possibile applicare la teoria delle PN Temporizzate e Colorate nel modello A-Nets, purchè all'interno di rappresentazioni parziali, come descritto nell'esempio precedente.

7. CONCLUSIONI

Nell'articolo viene presentato un approccio ibrido al problema della rappresentazione di azioni nel tempo. In particolare, è stato definito un ambiente in cui vengono integrati differenti approcci e metodologie: rappresentazione a reti semantiche per quanto riguarda l'aspetto logico dei problemi, rappresentazione mediante reti di Petri per quanto riguarda l'aspetto temporale, infine rappresentazione analogica di processi continui come in approcci di tipo qualitativo.
Inoltre il sistema è in grado di operare a diversi livelli di astrazione, a seconda delle esigenze della rappresentazione.

La prosecuzione della ricerca è orientata essenzialmente verso una maggiore utilizzazione della teoria delle reti di Petri allo scopo di ricavare proprietà di azioni da operazioni di analisi delle reti corrispondenti.

Un sistema basato su A-Nets è stato implementato utilizzando il linguaggio KEE [15] su Lisp machine Explorer [22], [3]. Tale sistema è stato progettato per applicazioni nel settore della robotica. Allo scopo di rappresentare conoscenza sul mondo reale, le funzionalità di KEE sono state integrate da un ambiente flessibile per modellare oggetti tridimensionali mediante octree e da un ambiente di generazione ed esecuzione di reti di Petri per la rappresentazione e la simulazione di processi ed eventi temporali.

Un secondo sistema basato su A-Nets, denominato Key-Music e sviluppato in ambiente Lisp/Flavors, è stato sviluppato per la rappresentazione di conoscenza musicale e per l'ausilio alla composizione [7].

8. RINGRAZIAMENTI

Desideriamo ringraziare Giovanni Adorni, Arrigo Frisiani e Renato Zaccaria per i preziosi suggerimenti e contributi.

La presente ricerca è stata supportata in parte da finanziamenti del Ministero della Pubblica Istruzione MPI 40% Robotica e in parte da finanziamenti nell'ambito del progetto EUREKA PROMETHEUS, sottoprogetto PRO-ART.

9. BIBLIOGRAFIA

[1] Adorni G., Gaglio S., Massone L., Morasso P., Zaccaria R., COGNITIVE MODELING OF PURPOSIVE ACTIONS, in Human Movement Understanding, P. Morasso and V. Tagliasco eds., North-Holland, 1986.

[2] Adorni G., Camurri A., Poggi A., Zaccaria R., INTEGRATING SPATIO-TEMPORAL KNOWLEDGE: A HYBRID APPROACH, Proceedings of the European Conference on Artificial Intelligence - ECAI 88, Munich, West Germany, 1988.

[3] Adorni G., Massone L., Poggi A., DEDALUS: A KNOWLEDGE BASED PLANNING SYSTEM,

Proc. 4th. Int. Conf. on Systems Research Informatics and Cybernetics, Baden Baden, West Germany, 1988.

[4] Ajmone Marsan M., Balbo G., Trivedi K. (Eds.): IEEE/ACM PROCS. INTERNATIONAL WORKSHOP ON TIMED PETRI NETS, Torino, Italy, 1985.

[5] Allen J.F., TOWARDS A GENERAL THEORY OF ACTION AND TIME, Artificial Intelligence 23, 123-154, 1984.

[6] Brachman R.J., Levesque H.J. (eds.), READINGS IN KNOWLEDGE REPRESENTATION, Morgan Kaufman Publishers, 1985.

[7] Camurri A., Giacomini M., Ponassi A., Zaccaria R.: KEY-MUSIC: AN EXPERT SYSTEM ENVIRONMENT FOR MUSIC COMPOSITION, Proc. International Computer Music Conference - ICMC 88, Koln, West Germany, 1988.

[8] Camurri A., Poggi A., Vercelli G., Zaccaria R., A-NETS: STRUCTURED REPRESENTATION OF TIME AND ACTIONS USING PETRI NETS, Proceedings of the Ninth European Workshop on Applications and Theory of Petri Nets, Venice, Italy, 1988.

[9] Camurri A. and Zaccaria R., AN EXPERIMENTAL APPROACH TO A HYBRID REPRESENTATION OF MUSICAL KNOWLEDGE, Proc. First International Workshop on AI and Music AIM-88, AAAI-88 (USA) and GMD, St. Augustin, West Germany, 1988.

[10] Fikes R.E., Nilsson N., STRIPS: A NEW APPROACH TO THE APPLICATION OF THEOREM PROVING TO PROBLEM SOLVING, Artificial Intelligence 2, 189-208, 1971.

[11] Fikes R., Rehler T., THE ROLE OF FRAME-BASED REPRESENTATION IN REASONING, Communications of the ACM 28(9), 904-920, 1985.

[12] Forbus K.D., QUALITATIVE PROCESS THEORY, Artificial Intelligence 24, 84-168, 1984.

[13] Goldberg A., Robson D., SMALLTALK-80: THE LANGUAGE AND ITS IMPLEMENTATION, Addison-Wesley, 1983.

[14] Hendrix G.G., MODELING SIMULTANEOUS ACTIONS AND CONTINUOUS PROCESSES, Artificial Intelligence 4, 145-180, 1973.

[15] Intellicorp, KEE: SOFTWARE DEVELOPMENT SYSTEM USER'S MANUAL, Version 2.1, Intellicorp, USA, 1985.

[16] Jensen K., COLOURED PETRI NETS, Lecture Notes in Computer Science, Vol.254, Brauer W., Reisig W., Rozenberg G. (eds.), Springer-Verlag, 248-299. 1987.

[17] McDermott D., A TEMPORAL LOGIC FOR REASONING ABOUT PROCESSES AND PLANS, Cognitive Science 6, 101-155, 1982.

[18] Minsky M., A FRAMEWORK FOR REPRESENTING KNOWLEDGE, Psychology of Computer Vision, P. Winston ed., McGraw-Hill, 1975.

[19] Moon D.A., OBJECT-ORIENTED PROGRAMMING WITH FLAVORS, SigPLAN Notice 21(11), 1986.

[20] Pagnoni A., STOCHASTIC NETS AND PERFORMANCE EVALUATION, Lecture Notes in Computer Science, Vol.254, Brauer W., Reisig W., Rozenberg G. (eds.), Springer-Verlag, 460-478, 1987.

[21] Peterson J.L., PETRI NET THEORY AND THE MODELING OF SYSTEMS, Prentice-Hall Inc., New York, 1981.

[22] Poggi A., DEFINIZIONE E SINTESI DI UN SISTEMA DI RAGIONAMENTO SPAZIO-TEMPORALE, Tesi di Laurea, DIST - Univ. di Genova, 1987.

LE CARATTERISTICHE DEI PRINCIPALI MODELLI SEMANTICI DI DATI

C. Tesauro
Istituto di Studi sulla Ricerca e la Documentazione Scientifica del CNR
Roma

RIASSUNTO

I modelli semantici di dati, definiti per rappresentare lo schema concettuale delle basi di dati, si sono evoluti fino a generare nuovi sistemi di gestione di basi di dati.
Sebbene i concetti fondamentali siano comuni, questi modelli vengono presentati utilizzando formalismi e termini differenti. All'origine di tali differenze vi sono le diverse esperienze tecnico-scientifiche degli autori.
Questo lavoro, partendo dalla definizione di una terminologia che consenta di esaminare le caratteristiche delle diverse proposte, presenta un'analisi dei più conosciuti modelli semantici dei dati, evidenziandone le caratteristiche fondamentali e le strutture statiche che definiscono.
In paricolare i modelli presi in esame sono: Entity-Relationship, Relational Model/Tasmania, TAXIS, Semantic Data Model, Functional Data Model, SAM*, SHM+, GALILEO.

ABSTRACT

Semantic data models, defined to represent the conceptual schema of data bases, were improved until the creation of new data bases management systems.
Altough there are common fundamental concepts, these models are presented with different formalisms and terms. The reason of such differences is in author's various technical and/or scientific experiences.This paper starts with the definition of a terminology useful to analyse the characteristics of the different models proposed. It shows an analysis of the most important semantic data models, showing their main characteristics and static structures.
In particular, the examined models are: Entity-Relationship, Smith & Smith, Relational Model/Tasmania, TAXIS, Semantic Data Model, Functional Data Model, SAM*, SHM+, GALILEO.

1. INTRODUZIONE

I modelli semantici di dati sono stati generati dalla necessità di superare le limitazioni progettuali imposte dai modelli relazionali, limitazioni riscontrabili soprattutto in fase di definizione dello schema concettuale. Il loro scopo fondamentale era di fornire degli pseudo-formalismi particolarmente utili per la loro economicità espressiva, flessibilità ed efficienza modellistica.
Rispetto ai modelli tradizionali, i Modelli Semantici si propongono di arricchire la struttura dello schema per poter rappresentare ulteriori aspetti del mondo reale, pur mantenendo la potenzialità espressiva dei DB relazionali. La principale caratteristica dei Modelli Semantici risiede, quindi, nei meccanismi di astrazione del modello.
In seguito, per alcuni di questi modelli, si è realizzato un prototipo funzionante con il duplice obiettivo di migliorare la flessibilità del modello definito (sono infatti frequenti i miglioramenti apportati ai modelli dopo una verifica realizzativa) e di consentire una rapida prototipazione dei sistemi progettati per verificare la loro

rispondenza ai problemi affrontati.

L'evoluzione dei Modelli Semantici è culminata con tentativi di utilizzare i prototipi realizzati sia per avviare un processo di automatizzazione della fase progettuale, mediante la creazione automatica di schemi interni, sia per gettare le basi di nuovi sistemi commerciali.

Questa evoluzione è dovuta anche all'integrazione di obiettivi comuni con altre aree di ricerca e sviluppo quali l'Intelligenza Artificiale (nel settore della Rappresentazione della Conoscenza) e dei Linguaggi, ed ha portato ad un aumento della complessità di questi modelli che, in alcuni casi, sono assurti al rango di veri e propri DBMS.

In questa breve rassegna sono raccolti alcuni dei lavori più significativi in questo contesto. Il metodo utilizzato è di confrontare le diverse proposte nel campo modellistico statico, tralasciando quindi sia gli aspetti relativi ai linguaggi sia quelli relativi alla dinamica. Per i motivi appena citati questa rassegna si differenzia da altri lavori sviluppati da recente sullo stesso argomento.

2. LA CARATTERIZZAZIONE DEI MODELLI SEMANTICI

In questo lavoro esiste un nucleo di termini, peraltro abbastanza ristretto, che necessita di una definizione precisa poichè non ne esiste ancora un uso uniforme generalmente accettato. Infatti avviene spesso che, in lavori diversi, gli stessi termini assumino significati differenti.

2.1 Terminologia

Il confronto delle diverse proposte è imperniato su alcuni concetti basilari.

2.2 Significato dei termini

I termini, e quindi i concetti fondamentali per questa analisi, si riferiscono a componenti elementari di un modello ed a relazioni esistenti tra di esse.

2.2.1 Identificazione dei componenti

Come componente di un modello possiamo avere:

a) Una rappresentazione di un elemento del mondo reale, che chiamiamo ISTANZA;

b) Una rappresentazione concettuale dell'istanza, ovvero un oggetto che consente di classificare gli elementi della realtà, ed allora parleremo di TIPO;

c) Le PROPRIETA', sono componenti che si ritrovano sia a livello di instanze, in cui assumono valori differenti, sia dei tipi, in cui hanno un significato concettuale e definizionale.

Esempio di istanza è "l'impiegato Mario Rossi", esempio di tipo è "impiegato", esempio di proprietà di istanza è "nome dell'impiegato = Mario", esempio di proprietà di tipo è "nome dell'impiegato".
Nell'analisi dei modelli le proprietà saranno considerate dopo le interrelazioni. Questa organizzazione del lavoro è stata scelta perchè spesso le proprietà hanno caratteristiche particolari riferite alle interrelazioni.

2.2.2 Le interrelazioni tra componenti

Le interrelazioni tra diverse componenti possono essere di varia natura, a seconda delle componenti da esse collegate. Avremo quindi come interrelazioni:

i) Relazioni tipo-istanza, come ad esempio la CLASSIFICAZIONE [1] che è il meccanismo che consente di definire un insieme di componenti come un unico componente di livello superiore, ovvero consentono la definizione degli oggetti concettuali come raggruppamenti di istanze omogenee. Un esempio di classificazione è l'interrelazione tra il Tipo "impiegato" e l'Istanza "l'impiegeto Mario Rossi".

ii) Relazioni tipo-proprietà (concettuale), che chiamiamo AGGREGAZIONE [2], è il meccanismo per cui interrelazioni tra componenti di livello inferiore, ad esempio la Proprietà, possono essere viste come elementi per la costruzione di componenti di livello superiore, come ad esempio il Tipo. Un esempio tipico di aggregazione è l'interrelazione tra il Tipo "impiegato" e le sue Proprietà che lo descrivono quali "nome", indirizzo ecc.

iii) Relazioni tipo-tipo, che sono relazioni gerarchi-

che, e che possono realizzare meccanismi di:

a) GENERALIZZAZIONE [3], che consente di vedere un insieme di oggetti simili come un unico oggetto generico, in tal caso è possibile definire un tipo T come astrazione generica di un tipo T1.

Un esempio classico di generalizzazione è quello di vedere i Tipi "docente" e "studente" come un unico oggetto generico rappresentato dal Tipo "persona".

b) COMPOSIZIONE, che consente di trasformare l'interrelazione tra componenti di livello superiore, come il Tipo, in un'unica componente anch'essa di livello superiore: è il caso in cui un tipo concorre alla definizione di un'altro.

Un esempio di composizione può essere la definizione del Tipo "corso" come composizione dei Tipi "docente", "studenti" e "sede" cui si aggiungono altri componenti come, ad esempio, la Proprietà "durata".

In Tabella 1 sono riportate le principali caratteristiche dei modelli esaminati in questa rassegna. Tali caratteristiche saranno approfondite nei paragrafi seguenti.

3. L'ENTITY-RELATIONSHIP MODEL

Basandosi sulla caratterizzazione dei modelli semantici fornita al paragrafo precedente, il modello E-R [4] non può essere definito un modello semantico in senso stretto. Tuttavia, il notevole contributo fornito dalla proposta originale sviluppata da Chen [4] alla nascita di tali modelli, rende necessario un suo esame.

Il modello E-R è stato sviluppato per sintetizzare i vantaggi offerti dai modelli dei dati reticolari e relazionali, proponendosi come elemento di sintesi ottenuto tramite un formalismo grafico dei dati da cui poter derivare schemi concettuali definiti secondo i modelli tradizionali.

3.1 Componenti

Nel modello E-R non viene esplicitato un concetto di Tipo come presentato in 2.2.2 ed in questo risiede il principale limite del modello. Esiste comunque una definizione implicita di tale concetto nella definizione degli Entity-set. Le istanze sono definite entità, mentre le proprietà sono dette attributi.

3.2 Interrelazioni

Le relazioni sono un elemento fondamentale del modello. La loro definizione risulta, però, sostanzialmente diversa da quelle degli altri modelli semantici, fondamentalmente a causa della mancanza della definizione del concetto di Tipo.

3.2.1 Classificazione

Le entità del modello vengono classificate in Entity-set se soddisfano un determinato predicato. L'Entity-set, inoltre, introduce parzialmente anche il concetto di Tipo, poichè le entità che lo compongono hanno alcune proprietà in comune.

3.2.2 Gerarchizzazione

Come per la classificazione, anche questo concetto risulta solo parzialmente definito a causa della mancata definizione del concetto di tipo. Infatti la struttura di

	E-R	RM/T	TAXIS	SDM	FDM	SAM*	SHM+	GALILEO
istanza	entità	entità	token	entità			oggetto	entità
tipo	entity-set	tipo-entità	classi	classi	entità	concetti	obj-class	classi
proprietà	attributi	proprietà	attributi	attributi	funzioni	attributi	proprietà	attributi
generalizz.	subset	sottotipi	IS-A	sottotipi	sottotipi	gen-entity	IS-A	sottotipi
composiz.	si	si	si	si	si	si	si	si
aggregaz.	si	si	no	no	si	si	si	no
tipo-istanza	si	si	si	si	no	no	si	si

Tab. 1: Schema riassuntivo delle principali caratteristiche dei modelli

insieme data alle classi consente di definire sottotipi come sottoinsiemi degli Entity-set. La struttura dei sottoinsiemi risulta però oscura poichè manca una esplicitazione dei meccanismi per la loro costruzione.

3.2.3 Composizione

La interrelazione composizionale nel modello E-R è definita come Relazione tra le entità. Le Relazioni sono classificate in Relationship Set. Come per le entità, non è chiaro se le relazioni definiscono componenti del modello e quale potrebbe essere la loro eventuale struttura.

3.2.4 Aggregazione

Anche in questo caso non esiste una definizione esplicita del concetto. Si può tuttavia dedurre la presenza di questa relazione dal concetto di classificazione associato agli Entity-set, classi di entità con alcuni attributi in comune.

3.3 Proprietà

Le proprietà sono dette attributi e sono definite come funzione che fa corrispondere ad una entità un insieme di valori.

3.4 La struttura risultante del modello

I meccanismi propri dei modelli semantici sono tutti presenti nell'E-R Model, sebbene le loro definizioni non siano particolarrnente approfondite. Si può quindi vedere come la struttura risultante del modello sia notevolmente articolata e permetta di modellare semplicemente strutture particolarmente complesse. Esistono però delle lacune, come la assenza di proprietà multivariate. Inoltre la definizione di Tipo renderebbe più potenti e più chiari alcuni dei meccanismi esistenti, come la Composizione.

4. RELATIONAL MODLEL/TASMANIA

Il Relational Model/Tasmania (RM/T) è una proposta di estensione del modello relazionale formulata da Codd [5] e si ricollega ai concetti base proposti da Smith e Smith [3]. Inoltre il modello RM/T si avvantaggia dell'esistenza di una versione realizzata che ha certamente fornito elementi per il completamento funzionale del modelo stesso.

I meccanismi di astratzione, Generalizzazione ed Aggregazione sono visti però in modo leggermente diverso dalla definizione data in 2.2.2, poichè se il loro significato è fondamentalmente lo stesso, è diversa la loro applicazione nel modello. Infatti, mentre la generalizzazione viene vista con differenze di carattere puramente implementativo, l'aggregazione viene definita in modo più completo.

4.1 Componenti

Codd definisce le istanze entità, ed utilizza quindi tale termine per denotare gli elementi della realtà. Il concetto di tipo, nel sistema RM/T, viene realizzato mediatnte l'oggetto denominato Tipo-entità. Il tipo-entità viene definito come classe di entità.

4.2 Interrelazioni

Nel modello RM/T, oltre alla disponibilità di tutte le interrelazioni presentate in 2.2.2, sono disponibili anche alcuni meccanismi particolari come la Generalizzazione Alternativa e la Cover Aggregation.

4.2.1 Classificazione

La classificazione è il meccanismo di denotazione del RM/T che consente la definizione dei tipi entità, suddividendoli in tre classi distinte:

l) Entità caratteristiche: sono quelle che cooperano alla definizione di altre entità realizzando quelle relazioni tipo- tipo di composizione illustrate in 2.2.2.. Possono essere viste come triple <E1, F, E2> dove E1 è il tipo-entità da definire, F la proprietà, ed E2 è l'entità corrispondente a F.

2) Entità associative: sono quelle che collegano tra loro entità di tipo diverso e si possono rappresentare come triple <A, E1, E2> in cui E1ed E2 rappresentano i due tipi-entità tra cui esiste l'associazione A;

3) Entità che non appartengono alle prime due classi: sono le cosiddette entità nucleo, definibili come coppie del tipo <E ,P> in cui E è il tipo-entità e P è l'insieme

delle sue proprietà.

4.2.2 Gerarchizzazione

Ad ogni istanza corrisponde un determinato tipo, ed in [5] è descritta la possibilità di generare sotto-tipi. Tale operazione può essere realizzata mediante due meccanismi, che sono:

l) Generalizzazione Incondizionata (GI) che realizza un meccanismo di inclusione incondizionata. Tale meccanismo consente, però, di definire un unico Tipo generico per un dato Tipo-entità.

2) Generalizzazione Alternativa (GA), che realizza un meccanismo di inclusione condizionata, in particolare permette di definire ulteriori Tipi generici per un dato Tipo-entità.

La GI ci riporta alla definizione citata in 2.2.2 mentre la GA fornisce un meccanismo che consente di descrivere la possibilità che esistano più Tipi che rappresentino una generalizzazione di un dato Tipo-Entità, come mostrato nell'esempio di Cliente [5]in fig.2.

Si evidenzia, inoltre, che il sottotipo eredita la appartenenza ad una delle predefinite classi del modello (caratteristiche, associative, di base).

4.2.3 Composizione

Le relazioni tipo-tipo di composizione possono essere definite in due modi:

l) Entità associative, che sono oggetti esistenti di per sè nel DB e manipolabili come entità qualsiasi.

2) Associazioni non-entità, sono semplici relazioni che intercorrono tra tipi-entità del DB, la cui esistenza è strettamente legata a quella di tutti i componenti da essa collegati.

Anche per le entità associative, però, si può legare la loro esistenza ai componenti collegati, inserendo un vincolo sulla non accettabilità di valori nulli in campi opportuni.

Altro meccanismo di composizione presente nel modello RM/T è la cosiddetta Cover Aggregation. Questa forma di composizione viene utilizzata per creare insiemi di istanze non omogenee rappresentandole come Tipi-entità. La differenziazione rispetto al meccanismi di composizione, quale quello mostrato in 2.2.2, di Cartesian Aggregation e di gerarchizzazione, si è resa necessaria poichè la Cover Aggregation considera le istanze e non i tipi o le proprietà e non può essere vista come generalizzazione.

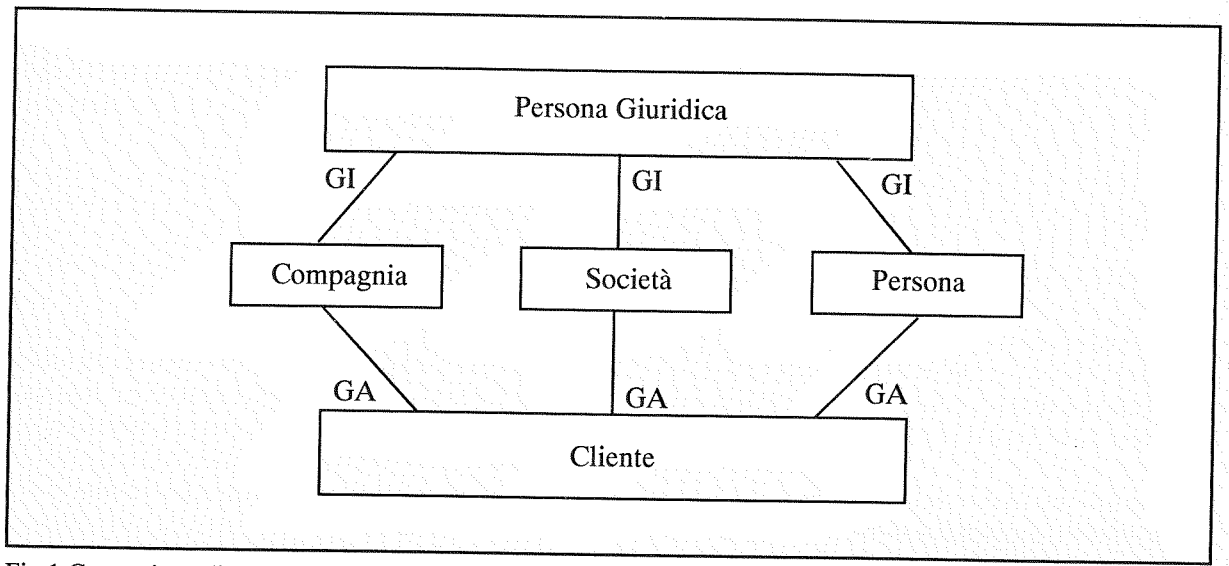

Fig.1 Costruzione di strutture gerarchiche complesse mediante i due meccanismi di gerarchizzazione disponibili

Un esempio di Cover Aggregation, presente in [5], può essere considerato il "Gruppo di Intervento" militare, che è costituito da istanze di Navi, Aerei, Carri arrnati e Soldati.

4.2.4 Aggrregazione

In [5] sono presenti due concetti di aggregazione. Il primo, analogo al concetto definito Smith e Smith, è detto Cartesian Aggregation. Essa può essere applicata a:

l) Proprietà dirette, o single-valued, per la costruzione di qualsiasi tipo-entità.

2) Entità caratteristiche, per la definizione di proprietà multi-valued di qualsiasi tipo-entità. Un esempio di entità caratteristica può essere la "data", vista come aggregazione di tre proprietà, giorno, mese ed anno.

3) Entità nucleo o associative, per la costruzione di entità associative.

4.3 Proprietà

In [5] sono strettamente collegate ai tipi entità e possono essere:

a) Proprietà dirette, sono sempre single-valued e sono quindi triple del tipo <E, p, V> in cui E è il tipo-entità, p la proprietà e V l'insieme dei suoi valori.

b) Proprietà indirette, che sono utilizzate per trattare le proprietà multi-valued. In questo caso si ricorre alle entità caratteristiche. La proprietà multi-valued viene quindi trasformata in sequenza di proprietà dirette di una entità caratteristica.

4.4 La struttura risultante del modello

Come evidenziato precedentemente, anche il modello RM/T trae beneficio dall'introduzione dei nuovi concetti di astrazione e la sua struttura risulta notevolmente diversa da quella dei DB relazionali tradizionali.

Gli elementi che caratterizzano la struttura del modello RM/T, ovvero la generalizzazione e la aggregazione, possono però avere anche caratteristiche diverse rispetto a quanto visto in 2.2.2.

5. IL MODELLO DI TAXIS

TAXIS [6] è un linguaggio definito per la costruzione di sistemi informativi interattivi. E' dotato di una struttura object-oriented e visto come componente principale di un ambiente programmativo. Tale ambiente contiene fra l'altro un DBMS relazionale in cui compilare le operazioni del linguaggio sulla base di dati.

Come nei casi precedenti anche per TAXIS il potenziamento del modello di rappresentazione dei dati è fornito dal concetto di astrazione. A differenza dei modelli precedentemente esaminati però TAXIS estrinseca il concetto di astrazione nella sola gerarchizzazione lasciando le relazioni tra oggetti come definite nei DBMS tradizionali.

Le gerarchie in TAXIS sono gestite in modo analogo a quelle delle reti semantiche e quindi sono di tipo Is-A. La relazione Is-A è una relazione di specializzazione, ovvero la forma gerarchica inversa della generalizzazione.
Il significato della relazione Is-A può essere descritto, in modo informale, come (A Is-A B) se tutte le istanze di A sono istanze di B. Ad esempio:

ADULTO IS-A PERSONA

specifica che ogni istanza di ADULTO è anche istanza di PERSONA.

La struttura derivante dall'uso dell'una o dell'altra forma gerarchica è comunque sostanzialmente la stessa.

5.1 Componenti

Vi sono tre tipi di componenti in TAXIS che sono:
l) Token; 2) Classi; 3) Metaclassi.

I token sono la rappresentazione delle istanze, ma in [6] sono definiti anche come rappresentazione delle costanti del linguaggio, siano esse stringhe o numeri.

Le classi sono collezioni di token che condividono proprietà comuni:
Ad esempio la classe IMPIEGATO, le cui istanze sono token come John Smith (che rappresenta una particola-

re persona) e NOME IMPIEGATO, le cui istanze sono token come la stringa John Smith.
Le classi in TAXIS rappresentano quindi il concetto di tipo. Le metaclassi sono componenti definiti in modo del tutto analogo alle classi con la differenza che collezionano classi invece di token. Ad esempio la metaclasse PERSONA può essre vista come contenente le classi IMPIGATO, STUDENTE, DOCENTE ecc.
Le proprietà sono dette attributi, come nei DB tradizionali.

5.2 Interrelazioni

Delle interrelazioni citate in 2.2.2, solo alcune sono presentate in modo esplicito per il modello di TAXIS. Il concetto di aggregazione non è esplicitato mentre la composizione è del tutto tralasciata.

5.2.1 Classificazione

Le classi, che in TAXIS rappresentano il concetto di tipo, sono suddivise, a seconda dei token contenuti, in

l) Variabili: sono le uniche contenenti token modificabili;

2)Aggregate: definite come prodotto incrociato delle estensioni delle loro proprietà sono modificabili solo cambiando l'estensione di una proprietà;

3) Definitivamente definite: sono classi la cui estensione è definita una volta per tutte al momento della loro creazione;

4) Test Definite: sono il caso più generale, e tutte le altre risultano una sua specializzazione. La loro estensione è definita dal risultato di una transazione.

Possiamo schematizzare le classi con quadruple del tipo: <N, K, AC, AP> in cui N è il nome e K la chiave. Per AC, Attributo-caratteristica, ed AD, Attributo-proprietà, bisogna sottolineare che TAXIS separa i concetti di attributo-caratteristica, inteso come proprietà intrinseca della classe, e attributo-proprietà, inteso come proprietà che può avere valore diverso in Istanze diverse.
Esiste inoltre un ulteriore elemento del linguaggio che è la metaclasse. Le metaclassi sono definite come classi di classi, ovvero come componenti del linguaggio che raccolgono classi con caratteristiche comuni. La loro schematizzazione è una tripla <N, C, AP> in cui N è il nome, C l'insieme delle classi ed AP l'insieme degli attributi-proprietà. Il ruolo degli attributi-proprietà è legato al concetto di astrazione della metaclasse. La proprietà di una metaclasse serve infatti a definire informazioni supplementari legate a più classi, ad esempio in una metaclasse PERSONA si può definire una proprietà ETA'-MEDIA che fornisce una informazione legata a più classi distinte come, ad esempio, DOCENTI, STUDENTI, IMPIEGATI ecc.

5.2.2 Gerarchizzazione

Questa relazione in TAXIS si ispira alla gerarchia Is-A delle reti semantiche, una relazione binaria che induce un ordinamento parziale. La relazione è applicabile indifferentemente a tutte le classi e metaclassi TAXIS. Il linguaggio prevede una possibile estremizzazione dell'uso di tale meccanismo, consentendo di costruire le classi e le metaclassi estreme di tali catene gerarchiche, che sono:

i) Le classi e le metaclassi più generali, con prefisso ANY, che conterranno tutti gli elementi della gerarchia;
ii) Le classi più specializzate, con prefisso NONE, e le metaclassi, con prefisso NO, sono per definizione vuote, poichè raggiungono un livello di specializzazione tale da non poter contenere alcun elemento.

Esistono poi degli altri vincoli che sono:

1) Di tipo estensionale: Se (C Is-A D), con C e D classi o metaclassi, allora un elemento di C è elemento di D;

2) Di tipo strutturale: Se (A Is-A B) e B è soggetto di una proprietà-attributo <(Cl,..,B,...Cn), p, D> allora esiste una proprietà-attributo <(Cl..,A,...Cn), p, E> e (E Is-A D). Ciò implica la esistenza di gerarchie anche tra le proprietà e i vincoli di integrità semantica

5.2.3 Composizione

Le relazioni tipo-tipo composizionali non sono incluse in TAXIS. In [6], però, viene citata la necessità della loro esistenza, specificando, inoltre, che la loro forma

piu opportuna è quella di un dato tipizzato, su cui siano definiti i relativi operatori.

5.2.4 Aggregazioni

Un concetto di aggregazione, che sia simile a quello descritto in 2.2.2, o a quelli del modello RM/T, non viene citato in modo esplicito. Tale meccanismo di astrazione è però concettualmente presente, altrimenti non esisterebbero classi e metaclassi.

5.3 Proprietà

In TAXIS sono chiamate attributi, e come già sottolineato in 5.2.1, sono previsti nelle forme di attributica ratteristica ed attributi-proprietà. La loro peculiarità risiede nella loro contemporanea presenza che introduce il concetto di definizione unica di proprietà comune. In [6] non sono, però, previste proprietà di tipo multi-valued.
Le proprietà sono definibili come triple e la loro funzione cambia a seconda dell'elemento cui è collegata, sia esso un token, una classe o una metaclasse.

La proprietà di una classe, ha un valore definizionale, potendo indurre il valore delle proprietà dei tokens di quella classe, ed è rappresentabile come <C, N, V> con C = (C1....Ci), i = 1,..., n, sono gli identificatori delle classi, N è il nome della proprietà e V l'insieme dei valori.
La proprietà di un token ha, invece, un significato puramente fattuale, ed è rappresentabile come <T, N, v> dove T è l'identificatore del token, N è il nome della proprietà e v valore ha assunto. La proprietà di una metaclasse è rappresentabile come una tripla del tipo <M, P, V> dove M è il nome della metaclasse, P è la sua proprietà e V l'insieme dei suoi valori.

5.4 La struttura risultante del modello

La mancanza del concetto di aggregazione visto nei paragrafi precedenti e delle proprietà multi-valued, semplifica notevolmente la struttura del modello adottato da TAXIS.

6. IL SEMANTIC DATA MODEL

Il Semantic Data Model [7] è un formalismo di descrizione e strutturazione di basi di dati che, come per i precedenti, ha come scopo fondamentale il maggior ampliamento possibile del significato dei dati stessi utilizzando gli attuali DBMS.
A differenza dei casi precedenti però, in [7] non si trattano esplicitamente aggregazione e specializzazione ma solo la Cover Aggregation, vista nel modello RM/T, e la categorizzazione.

La categorizzazione è una forma di astrazione originale rispetto ai modelli sin qui esaminati che consente di definire raggruppamenti di oggetti che hanno valori uguali per determinate proprietà. Tale nuova forma di astrazione sembra definita in funzione di un tentativo di utilizzare il modello per la definizione di un prototipo di sistema informativo.

Un esempio di categorizzazione può essere rappresentato come la possibilità di definire un oggetto MODELLO DI NAVE che raggruppa tutte le istanze di NAVE con uguale valore dell'attributo MODELLO.
Aggregazione cartesiana e specializzazione sono comunque presenti nella struttura del modello anche se non esplicitamente evidenziate.

6.1 Componenti

Le istanze dell'SDM sono definite come entità, mentre i tipi sono rappresentati dalle classi. Le proprietà sono dette attributi.

6.2 Interrelazioni

L'insieme delle relazioni definibili nel modello SDM copre completamente la definizione data in 2.2.2 con l'aggiunta di un ulteriore caso, quello della Categorizzazione.

6.2.1 Classificazione

Le entità sono, quindi, organizzate in classi, e sono dotate delle seguenti caratteristiche:
1) Hanno un nome; 2) I loro elementi sono entità; 3) Ospitano, opzionalmente, una descrizione in linguaggio naturale; 4) Hanno delle proprietà, dette attributi;
5) Possono essere di base o non-base. Nell'ultimo caso, se la classe è di base, allora:
i) Ha associata una lista di attributi elemento o chiavi di

identificazione dell'elemento;

ii) La presenza di duplicati è opzionale.

Una loro formalizzazione può essere quindi con coppie, per le classi non-base, del tipo <C, P> in cui C è il nome della classe e P l'insieme delle proprietà, e triple, per le classi di base, del tipo <C, K, P> in cui K è la chiave di accesso alle entità.

Le classi possono essere viste come raccolte di: a) Oggetti reali; b) Eventi; c) Entità non omogenee (per realizzare una Cover Aggregation, e categorizzazioni, che rappresentano una ulteriore tipizzazione delle entità); d) Nomi o identificatori sintattici.
L'SDM prevede inoltre l'esistenza di classi predefinite, come ad esempio la classe STRING.

6.2.2 Gerarchizzazione

La descrizione data in [7] delle relazioni tipo-tipo gerarchiche è piuttosto semplice. Tali relazioni sono infatti definite mediante il meccanismo di definizione di sottotipo che introducono il concetto di gerarchia visto in 2.2.2. La generazione del sottotipo avviene mediante l'applicazione di predicati di vario genere.

6.2.3 Composizione

Le relazioni tipo-tipo composizionali sono contenute nella possibilità di definire la proprietà di una classe utilizzando un'altra classe.

6.2.4 Aggregazione

Nel modello SDM esiste la possibilità di trattare quella aggregazione che in [5] viene chiamata Cover Aggregation. Esiste infatti una connessione tipo-tipo di raggruppamento, definita come classe del second'ordine. Tale classe del second'ordine viene definita, con un meccanismo analogo alla generazione del sottotipo, mediante l'applicazione di predicati del tipo "where specified" che agiscono su valori appositi gestiti esplicitamente dagli utenti. Ad esempio, nel caso di un convoglio di navi, esisterà una proprietà della classe NAVE il cui nome è INSERITA IN CONVOGLIO ed il cui valore sarà uguale per tutte le navi di un dato convoglio.

6.3 Proprietà

Nel modello SDM le proprietà, dette attributi sono caratterizzate da: 1) Nome; 2) Valore, che può essere anche un'entità o una classe; 3) Applicabilità i specifica, possono essere infatti di entità o di classe; 4) Descrizione opzionale ospitata; 5) Possono essere single o multi-valued; 6) Possono essere mandatory; 7) Non modificabilità, laddove non esplicitamente dichiarato; 8) Se sono multi-valued, sono non overlapping; 9) Possono essere collegate ad altre proprietà.

Le proprietà possono essere appresentate come triple <C, P, V> in cui C è la classe, P la proprietà e V l'insieme dei valori, nel caso sia o riferite a classi, o come <E, P, v> in cui E è l'entità e v il valore della proprietà P, se riferite ad entità.

6.4 La struttura risultante del modello

La struttura del modello risulta ancora più complessa e completa di quella dei modelli esaminati in precedenza. Lo SDM, infatti, dispone dei tre meccanismi di astrazione comuni ai modelli sin qui esaminati ed in più dispone del meccanismo di categorizzazione che introduce nuove potenzialità descrittive, nate probabilmente dalle esigenze sorte nell'uso di una versione implementata di SDM.

Per quanto riguarda la specializzazione e la aggregazione non vi sono diversificazioni nelle potenzialità descrittive rispetto alle proposte precedentemente esaminate.

7. FUNCTIONAL DATA MODEL

Il Functional Data Model (FDM) è il modello su cui si basa DAPLEX [8]. DAPLEX rappresenta un ulteriore tentativo di definire un linguaggio di programmazione per sistemi di basi di dati. Il modello FDM è basato su due costrutti fondamentali, entità e funzioni, che modellano gli oggetti della realtà, le loro proprietà, e le loro interrelazioni.
Mentre il concetto di entità utilizzato nel modello FDM è analogo a quello presentato in 2.2.2, il concetto di funzione risulta completamente nuovo. Le funzioni, infatti, assolvono il compito di gestire sia le proprietà che le interrelazioni. Ad esempio, la funzione Nome,

crea un legame tra entità di tipo STUDENTE ed entità di tipo STRINGA, così come la funzione Dip crea un legame tra entità di tipo STUDENTE ed entità di tipo DIPARTIMENTO.

7.1 Componenti

I tipi sono chiamati entità e possono essere rappresentati come coppie <N, P> in cui N è il nome dell'entità, e P è l'insieme delle proprietà. Non esiste alcun riferimento a proprietà chiave o a distinzioni tra le proprietà.

7.2 Interrelazioni

Le relazioni del modello FDM sono gestite tramite le funzioni. Ciò implica una definizione completamente diversa da quanto visto nei modelli precedenti. L'ottica con cui vengono esaminate risulta, quindi, legata alle funzionalità piuttosto che alla definizione stessa.

7.2.1 Classificazione

In FDM la relazione tipo-istanza detta classificazione non è esplicitata. La si può ritenere presente al livello implicito, ma non compare alcun riferimento di ciò in [8].

7.2.2 Gerarchizzazione

Viene realizzata applicando alle entità il meccanismo dei sottotipi, ottenendo i tipi Superentity e Subentity.

La subentity può essere definita come:

1) Sottoinsieme: subentity definita mediante la applicazione di un predicato ad una proprietà gestita appositamente dall'utente;

2) Partizione: caso simile a quello del sottoinsieme, con la differenza che i suoi elementi non possono appartenere ad altri sottoinsiemi della stessa partizione.

3) Restrizione: subentity definita mediante la applicazione di predicati su qualsiasi proprietà della classe.

7.2.3 Composizione

Da quanto descritto in [8], le relazioni di composizione sono presenti in DAPLEX, come definite in 2.2.2. Tuttavia la mancata definizione di una classificazione rende oscura la sua realizzazione.

7.2.4 Aggregazione

E'presente, in forma analoga a quella definita in 2.2.2, nella definizione di entità. Il caso della Cover Aggregation, invece, viene trattato in FDM con modalità notevolmente differenti da quelle prospettate negli altri modelli citati in questa rassegna. L'aggregazione, infatti, non viene trattata come entità, ma realizzata a livello di DML mediante opportuni programmi. Ciò ricorda molto la soluzione proposta per tale problema dai DB classici.

7.3 Proprietà

In FDM le proprietà sono definite come funzioni e direttamente legate alle entità. Sono previste sia le proprietà dirette che quelle indirette.

7.4 La struttura risultante del modello

Come evidenziato, quindi, FDM modella la realtà utilizzando due costrutti: Entità e funzioni. Tuttavia sono presenti in FDM, sebbene non esplicitamente evidenziati, i costrutti propri dei modelli semantici di dati, per cui la struttura complessiva dei dati di FDM risulta quasi analoga a quella degli altri modelli.

8. SEMANTIC ASSOCIATION MODEL (SAM*)

Il modello SAM* [9] si differenzia in modo sostanziale da tutti gli altri modelli presentati in questa rassegna. Tra le sue caratteristiche peculiari vi è un maggior legame con la fase di progettazione concettuale, che si estrinseca in un disimpegno dalla rappresentazione delle istanze. Questa funzionalità risulta infatti delegata alle cosiddette G-relations, ovvero relazioni simili a quelle dei DBMS tradizionali.

Altra caratteristica peculiare è la forte tipizzazione del modello, analoga a quella dei linguaggi algoritmici, che si esprime in modo particolare nella definizione delle proprietà.

Il tipo di dato è la caratterizzazione di un insieme di

valori e di un insieme di operazioni che sono applicabili a quei valori. Esistono tipi predefiniti, come gli Interi, i Reali, le Stringhe ecc. e tipi definiti dall'utente come, ad esempio, l'orario che è composto da ora, un intero compreso tra 0 e 23, minuto, intero compreso tra 0 e 60, e secondo, intero compreso tra 0 e 60.

Ulteriore differenza caratterisrica rispetto agli altri modelli presentati, che discende direttamente dalle precedenti, è nella definizione dei componenti fondamentali. Infatti come elementi base del modello sono definiti i concetti e le associazioni, dove per associazione si intende una interrelazione tra concetti mentre i concetti possono essere:

a) Atomici, ovvero rappresentazioni di oggetti fisici o astratti, eventi, o qualsiasi elemento della base di dati che possa essere definito come non decomponibile;

b) Non-Atomici, ovvero rappresentazioni di elementi che possono essere definiti in termini di altri elementi, siano essi atomici o meno.

8.1 Componenti

Sebbene sostanzialmente diverso dagli altri, il modello SAM* può comunque essere presentato in modo analogo. La assenza di una trattazione specifica del concetto di istanza è stata già evidenziata. Esistono invece un concetto di Tipo che può essere assimilato al Concetto non-atomico nel modello SAM*, ed un concetto di proprietà detta attributo.

8.2 Interrelazioni

A differenza degli altri modelli, in SAM* esistono sette interrelazioni, dette associazioni, tra le componenti. Alcune di esse sono particolarizzazioni delle relazioni definite in 2.2.2 e saranno esaminate insieme.
Caratteristica peculiare delle associazioni nel modello SAM* è che ciascuna di esse da origine ad un tipo specifico.

8.2.1 Classificazione

Per come sono state gestite le istanze, non esiste una interrelazione Tipo-Istanza quale la classificazione.

Esiste però un altra forma di relazione Tipo-istanza detta Associazione di Sommarizzazione. Questa relazione consente di definire un oggetto che per funzionalità è simile alla metaclasse di Taxis, offrendo maggiori potenzialità.

Questo oggetto, infatti, consente di definire proprietà caratteristiche dell'insieme delle eventuali istanze come, ad esempio, valori medi delle proprietà di entità, ed utilizzato con l'Associazione di Prodotto-Incrociato, descritta in seguito, consente una rappresentazione di oggetti complessi particolarmente utili nelle applicazioni statistiche.

8.2.2 Gerarchizzazione

Il concetto di generalizzazione in SAM* è espresso in modo analogo allo standard dei modelli semantici. Il concetto generico è definito Entità Generica e gli elementi in esso riuniti sono detti Occorrenze Generiche.

8.2.3 Composizione

La composizione è definita in duplice modo. Esiste un concetto di composizione, analogo al concetto proposto da Smith e Smith [3] con il nome di aggregazione (4.2.3), detto Associazione di Interazione. Gli elementi così definiti vengono chiamati Tipo Relazione. Esiste poi una Associazione di Composizione che definisce un concetto costituito da più concetti componenti e che viene detto Tipo Composizione. La distinzione tra queste due definizioni non è molto chiara. Sembra che la differenza risieda nel tipo dei componenti che nel primo caso, Interazione, possono essere Aggregati o Attributi, mentre nel secondo caso, Composizione, possono essere di qualsiasi tipo. Ciò però comporta una sovrapposizione tra il concetto di Interazione ed di Aggregazione, la cui differenziazione è puramente operativa.

8.2.4 Aggregazione

Anche l'aggregazione, come la composizione, è definita in duplice modo. Nel primo caso, Associazione di Aggregazione, un concetto descritto da più concetti componenti, detti Occorrenze di Entità, viene detto Tipo Entità. Un Tipo Entità può essere definito come aggregazione di attributi e/o di altri Tipi Entità, creando

così una sorta di gerarchia di aggregazioni. Nel secondo caso, Associazione di Prodotto-Incrociato, l'oggetto definito, detto Tipo Prodotto-Incrociato, differisce dal Tipo Entità esclusivamente per il fatto che può essere identificato esclusivamente da tutto l'insieme dei concetti componenti e non solo da un sottoinsieme di essi, come invece avviene per il Tipo Entità.

8.3 Proprietà

Le proprietà sono definite da una Associazione di Concetti Atomici, detta Appartenenza, e sono denominate Tipo Attributo. Le proprietà del modello SAM* possono quindi essere composte.

8.4 La struttura risultante del modello

La definizione strutturale del modello SAM* [9] è indubbiamente la più complessa tra quelle presentate in questa rassegna. Tale complessità non necessariamente si riflette in una maggiore potenzialità descrittiva. La struttura risultante non appare, infatti, sostanzialmente diversa da quella definita dagli altri modelli semantici. Inoltre, l'assenza di un meccanismo di classificazione vero e proprio e di una caratterizzazione delle istanze complicano indubbiamente l'opera di descrizione di una base di dati.
Altrettanto si può dire per quelle relazioni in più che sono definite nel modello SAM*, che sono di sicura efficacia in deterrninate situazioni, ma possono confondere le idee del progettista anzichè semplificarle.

9. EXTENDED SEMANTIC HIERARCHY MODEL (SHM+) N

Il modello SHM+ [1] è stato concepito in un contesto di definizione di tecniche per la progettazione e la specifica di applicazioni su basi di dati. In quel contesto si esaminano i problemi della progettazione e della specifica sia degli aspetti statici che degli aspetti dinamici. Il modello SHM+ è basato, come gli altri, sulla estensione del concetto di astrazione secondo le direttrici evidenziate al paragrafo 2.
Elemento caratterizzante è il costante riferimento alle tecniche di Rappresentazione della Conoscenza tipiche dell'ambito dell'Intelligenza Artificiale che evidenzia ulteriormente lo stretto legame tra questi due indirizzi di ricerca.

9.1 Componenti

Nel modello SHM+ è definito come unico componente strutturale, l'oggetto, che corrisponde a ciò che in 2.2.2 è stato chiamato istanza. Il concetto di Tipo è definito come Object Class, ma sebbene non esplicitamente definito, è considerato ed utilizzato come componente strutturale. La definizione di proprietà coincide con quella presentata in 2.2.2.

9.2 Interrelazioni

Le relazioni costituiscono la seconda componente fondamentale del modello SHM+, ma alcune di esse sono viste in modo sostanzialmente diverso da quanto mostrato nel paragrafo 2. Va notato come le relazioni siano definite esplicitamente dagli autori come rappresentazione delle relazioni tipiche del settore delle Basi di Conoscenza.

9.2.1 Classificazione

Questo concetto è definito in modo classico, ovvero come associazione tra le istanze, dette oggetti, ed il concetto di tipo, detto Object Class, definito come relazione Instance Of tipica delle Basi di Conoscenza. Non esiste una caratterizzazione specifica delle classi.

9.2.2 Gerarchizzazione

Anche la gerarchizzazione è definita in modo analogo a quanto riportato in 2.2.2, ed anche in questo caso la interrelazione è descritta come la relazione tipica delle Basi di Conoscenza, ovvero la Is-A.

9.2.3 Composizione

La interrelazione di composizione tra due tipi nel modello SHM+ è definita associazione ed è descritta come la relazione Member Of delle Basi di Conoscenza.

9.2.4 Aggregazione

Anche la definizione di aggregazione, come quelle delle altre interrelazioni, è in perfetta sintonia con quanto mostrato in 2.2.2, ed è descritta come la relazione Part Of delle Basi di Conoscenza.

9.3 Proprietà

Anche in questo caso non esiste una trattazione specifica del problema. Si intuisce la loro presenza, ma non vengono esplicitate descrizioni e caratteristiche specifiche delle proprietà.

9.4 La struttura risultante del modello

L'assenza di descrizioni specifiche esplicite dei componenti e delle loro interrelazioni consente di avere una visione solo generica della struttura complessiva del modello SHM+. Ciò che si può evidenziare è la presenza di tutti i meccanismi classici dei modelli semantici che conferiscono al modello le usuali potenzialità descrittive.

10. IL MODELLO DI GALILEO

GALILEO [10] è un linguaggio per applicazioni su basi di dati definito per rendere disponibili i meccanismi di astrazione sia dei modelli semantici per basi di dati che dei più recenti linguaggi di programmazione.
Gli elementi che GALILEO eredita dai modelli semantici sono la aggregazione e la specializzazione mentre dai linguaggi, ed in particolare dalla famiglia dei cosiddetti Pascal-like, eredita il meccanismo dei tipi di dati, già analizzato nel caso di SAM*.

10.1 Componenti

In GALILEO le istanze sono dette entità, mentre l'introduzione del livello intensionale, ovvero la definizione di tipo, è contenuta nelle classi, che sono oggetti fortemente tipizzati nel senso linguistico del termine. Le proprietà, dette attributi, non sono trattate in modo esplicito.

10.2 Interrelazioni

Anche per le relazioni non vi sono riferimenti espliciti per alcuni casi. Esistono, comunque, sicuramente sia le relazioni tipo-tipo, gerarchiche e composizionali, sia la Cover Aggregation.

10.2.1 Classificazione

Anche in GALILEO esiste una schematizzazione delle classi, legata al concetto di tipizzazione già sottolineato precedentemente. Avremo quindi che esistono: 1) Classi base, che consentono di definire la struttura degli oggetti. 2) Sottoclassi, che sono classi definibili mediante derivazione, dalle classi base.
Una possibile schematizzazione delle classi di Galileo è quella di quadruple <N, PN, PM, K> dove N è il nome della classe, PN sono le proprietà non identificabili, PM le proprietà modificabili e K la chiave di accesso.

10.2.2 Gerarchizzazione

Le relazioni tipo-tipo gerarchiche realizzano una vera e propria gerarchia Is-A che, come già visto, costituisce una relazione binaria con ordinamento parziale. Sono presenti per il meccanismo delle sottoclassi. Le sottoclassi sono definite in modo analogo alla definizione di subentity in 7.2.2.
Le sottoclassi, inoltre, possono utilizzare gli operatori già definiti per le classi, nel caso in cui trattino proprietà non modificate. Altrimenti si possono definire operatori derivati adattati alla particolare sottoclasse. Tutto questo denota la presenza di una tipizzazione degli operatori che, in questo caso, vengono trattati in modo analogo ai dati.

10.2.3 Composizione

Tale relazione viene realizzata consentendo la definizione di una classe come proprietà di un'altra classe. Trattandosi di una caratteristica delle proprietà, sarà evidenziata in 10.3.

10.2.4 Aggregazione

L'aggregazione viene vista in GALILEO come associazione di proprietà che concorrono alla definizione di un oggetto. Rispetto ai DBMS tradizionali, GALILEO offre in più possibilità di utilizzare la rappresentazione di un oggetto come attributo della rappresentazione di un altro oggetto. Non esiste invece alcun riferimento ai concetti di aggregazione presentati in [3], e [5]

10. 3 Proprietà

Sono definite attributi, ed in [10] non sono trattate esplicitamente. Si può dedurre, come già evidenziato, che gli attributi di una classe possono essere a) Chiavi

b) Costanti; C) Varabili; d) Opzionali; e) Derived.

Una loro possibile formalizzazione è quella di triple <C, A, V> in cui C è il nome della classe, A è il nome dell'attributo e V il suo insieme di valori.
Si sa inoltre, che possono essere single o multi-valued. Il caso del multi-valued può essere trattato in due modi: i) Definendo come tipo dell'attributoun tipo di dato astratto che riunisce più attributi; ii) Definrndo come tipo dell'attributo un oggetto di tipo classe.

Nel secondo caso avremo, ovviamente, attributi indiretti. Gli attributi, inoltre, devono rispettare i seguenti vincoli:

I) La modifica dei valori è consentita esclusivamente in caso di attributi definiti variabili.

II) Possono essere definiti attributi opzionali, i cui valori non devono essere specificati immediatamente.

III) Possono essere definiti attributi derivati, il cui valore è funzione di altri attributi del modello.

10.4 La struttura risultante del modello

Ai fini della struttura del modello, la specializzazione risultata quindi il principale meccanismo di strutturazione che incrementi le potenzialità descrittive rispetto ai DBMS tradizionali. Il meccanismo della specializzazione in GALILEO è definito utilizzando la relazione Is-A.

11. CONCLUSIONI

Come si è potuto osservare, le strutture definite dai diversi modelli sono fondamentalmente analoghe, poichè tutti i modelli risultano dotati di una serie di meccanismi di base comuni, come evidenzia la Tab. 1.

Le differenze riscontrabili, sorte fondamentalmente dall'applicazione dei singoli prototipi a problematiche differenti, fomiscono meccanismi che risultano particolarrnente utili in casi particolari.

Per questo motivo il progettista che volesse sfruttare pienamente le potenzialità di un modello semantico per definire una base di dati, è costretto ad effettuare una scelta ben ponderata che potrebbe semplificare notevolmente la fase successiva del lavoro.

12. BIBLIOGRAFIA

[1] Brodie M. L., ON THE DEVELOPMENT OF DATA MODELS, da On Conceptual Modelling, Springer Verlag 1984.

[2] Peckham J., Maryanski F., SEMANTIC DATA MODELS, ACM Computing Surveis, Vol.20, N. 3, Settembre 1988.

[3] Smith J. M., Smith D.C.P., DATABASE ABSTRACTION: AGGREGATION AND GENERALIZATION, ACM ToDS, Vol.2, N.2, Giugno 1977.

[4] Chen P.P., THE ENTITY - RELATIONSHIP MODEL- TOWARD A UNIFIED VIEW OF DATA, ACM ToDS, Vol.1, N., Marzo 1976.

[5] Code E. F., EXTENDING THE DATA BASE RELATIONAL MODEL TO CAPTURE MORE MEANING, ACM ToDS, Vol.4, N.4, Dicembre 1979.

[6] Mylopulos J., Bernstein P.A., Wong H.K.T., A LANGUAGE FACILITY FOR DESIGNING DATABASE INTENSIVE APPLICATIONS, ACM ToDS, Vol.5, N.2, Giugno 1980.

[7] Hammer M., Mcleod D., DATA BASE DESCRIPTION WITH SMD: A SEMATIC DATA MODEL, ACM Computing Surveis, Vol.6, N. 3, Settembre 1981.

[8] Shipman D.W., THE FUNCIONAL DATA MODEL AND THE DATA LANGUAGE DAPLEX, ACM ToDS, Vol.6, N1., Marzo 1981.

[9] Su S. Y.M.: Sam* A SEMANTIC ASSOCIATION MODEL FOR CORPORATE AND SCIENTIFIC DATABASES, Information Sciences, Vol. 29, 1983.

[10] Albano A., Cardelli L., Orsini R., GALILEO: A STRONGLY - TYPED, INTERACTIVE CONCEPTUAL LANGUAGE, ACM ToDS, Vol.10, N.2, Giugno 1985.

S.I.S.M.A.: UN ELABORATORE SISTOLICO RICONFIGURABILE A RUNTIME

Giuseppe Rossano (*,**), Roberto Gabaglio (*,**), Daniela Ghiroldi (*,**),
Roberto Grande (*), Giulio Barbaglia (**)
(*) Dipartimento Scienze dell'Informazione Università di Milano
(**) Montedison Sistemi d'Automazione.

RIASSUNTO

Il progetto SISMA prevede la realizzazione di una workstation sistolica da utilizzare in applicazioni pratiche; si é perciò inserito in un ambiente di computazione tradizionale un coprocessore sistolico modulare, riconfigurabile a runtime. L'articolo delinea le caratteristiche di tale workstation e descrive sia il prototipo hardware che l'ambiente di sviluppo software realizzati.

ABSTRACT

SISMA project developes a systolic workstation to use in real application; SISMA Computer is a modular systolic, runtime modifiable architecture coprocessor inserted in a traditional computation environment. The paper presents the features of the workstation; it also describes the hardware prototype and the software developement environment realized.

1. INTRODUZIONE

I progressi tecnologici si riflettono in due modi sulle performances dei calcolatori: direttamente mettendo a disposizione componenti HW più veloci, indirettamente attraverso la loro influenza sulle architetture. La tecnologia VLSI ha prodotto notevoli incrementi di prestazione per ciò che riguarda velocità e livello di integrazione dei componenti; tuttavia le sue potenzialità non sono state completamente sfruttate a causa dei limiti delle correnti metodologie progettuali. Il VLSI é infatti un environment tecnologico che richiede idee nuove in termini di architetture di computers, modelli computazionali e metodologie di progettazione. Sotto questo punto di vista gli array sistolici costituiscono un'interessante innovazione [1]. In un array di processori sistolici ogni PE (Processor Element) esegue operazioni molto semplici, la comunicazione tra i processori é regolare e locale, la comunicazione con il mondo esterno avviene solo attraverso i PE periferici; i dati fluiscono ritmicamente dalla memoria centrale (o da una qualsiasi device estema) attraverso ogni Processor Element prima che i risultati tomino in memoria; si ha così un utilizzo multiplo ed esaustivo di ogni dato. Perchè ciò avvenga nell'array di processori sistolici deve esistere una quantità di memoria (distribuita) - che non dipende dalla dimensione dell'array ma solo dalla natura dell'algoritmo sufficiente a trattenere i dati per il tempo necessario alla computazione.
Infine l'array di PE può essere esteso in modo modulare per soddisfare richieste di performances variabili.
Una generale classificazione permette di suddividere le architetture sistoliche in algorithmic oriented e programmabili. Le prime sono caratterizzate da prestazioni elevate, ma hanno un limitato campo di applicazione;

mentre le seconde, pur conservando una buona efficienza, presentano un maggiore grado di flessibilità. Un ulteriore grado di flessibilità si ottiene se si predispone l'architettura ad essere riconfigurata [2].

Con il progetto S.I.S.M.A. (Systolic Integrated Software Modifiable Architecture) si é voluta realizzare una architettura sistolica dotata di entrambe le caratteristiche di programmabilità e riconfigurabilità (sia dimensionale che strutturale) da inserire in un ambiente di computazione tradizionale [3].
Ne risulta un ambiente, la workstation sistolica, composto da un host computer, da uno o più computer sistolici (SISMA Computer o SISMA-C), comunicanti e utilizzati come coprocessori, e da opportune periferiche di input/output. Ogni SISMA Computer é espandibile modularmente, connettendosi con altri SISMA-C in orizzontale (fig. 1). L'insieme dei SISMA-C sarà dedicato all'esecuzione di subroutines particolarmente onerose dal punto di vista computazionale e che richiedono elaborazioni parallele su matrici di dati.
In fase di progettazione si era previsto che SISMA-C fosse composta da due sottounità principali: un controller e l'unità di elaborazione SISMA contenente l'array sistolico vero e proprio. Durante la realizzazione del prototipo qui descritto si é preferito demandare le funzionalità del controller all'host, ottenendo così una più rapida prototipizzazione ed una maggiore flessibilità nel testing della parte centrale del progetto, ma compromettendo il raggiungimento delle massime prestazioni teoriche.

2. G.A.P.P.

Il building block dell'unità di elaborazione SISMA é il chip N.C.R. GAPP (Geometric Arithmetic Parallel Processor), uno dei primi circuiti integrati della classe Single Instrucuon Multiple Data Stream (SIMD) [4]. GAPP é un processore sistolico di tipo programmabile realizzato in tecnologia CMOS con scala di integrazione VLSI.

In ciascun chip sono presenti 72 Processor Element, organizzati in una matrice 6x12; ognuno di essi é un processore a singolo bit composto da una ALU, 128 bit di RAM e 4 registri da un bit: tre (NS, EW, C) sono in input alla ALU ed il quarto (CM) é dedicato all'I/O.
La struttura del GAPP consente lo scambio di dati tra PE adiacenti nelle 4 direzioni; il medesimo schema di comunicazione mesh-connected si attua tra i PE di frontiera dei chip direttamente interfacciati. L'architettura di GAPP rende così possibile costruire array modulari di dimensioni arbitrarie, il cui comportamento ricalca quello del chip di base. Tutti i PE eseguono sincronamente e concorrentemente la stessa istruzione operando su dati privati. L'instruction set del chip GAPP comprende istruzioni composte da 5 campi che pilotano in parallelo i registri e la RAM di tutti i PE.

All'interno di ogni PE la RAM é vista come una colonna di 128 bit indirizzabili separatamente. I dati nell'array GAPP sono orgánizzati in variabili di tipo immagine, il cui valore è rappresentato da piani sovrapposti di bit; ogni piano è costituito dai bit di RAM di tutti i processori identificati dal medesimo indirizzo. Poiché la particolare struttura del GAPP manipola dati in formato bit-serial/word-parallel, è necessaria una loro trasformazione dal tradizionale formato word-serial/bit-parallel. (fig. 2)

Infine vi è un segnale di output, globale a tutto l'array (Global Output), costituito dal NOR tra il valore di tutti i registri NS; tale segnale fornisce un'informazione relativa allo stato della computazione in tutto l'array.

3. CARATTERISTICHE FUNZIONALI

Come già accennato, la workstation sistolica è composta da un host, da uno o più coprocessori SISMA-C e dalle periferiche; l'utente interagisce con SISMA-C tramite un insieme di strumenti SW, residenti sull'host, che ne facilitano la programmazione e l'accesso. In un tipico schema di utenza l'utente richiede all'host l'esecuzione di un programma, scritto in linguaggio ad alto livello (ad esempio il C), all'interno del quale sono presenti chiamate a particolari subroutines delle quali è demandata l'esecuzione ai coprocessori SISMA-C. Il codice relativo a queste subroutines, scritto in linguaggio ad alto livello (SISMA Language o SL), viene tradotto in una sequenza di istruzioni a basso livello ed inviato ai SISMA-C.

Le diverse unità SISMA-C possono essere viste dal programmatore come un'entità monolitica in modalità SIMD, oppure come computer distinti in modalità MultiSIMD; nel primo caso tutti i SISMA-C eseguono

il medesimo codice e il flusso dei dati tra processori adiacenti avviene come se esistesse un'unica unità; nel secondo caso si hanno più SISMA Computer indipendenti ognuno dei quali esegue un codice privato.
La definizione della modalità di funzionamento non è statica, bensì è possibile pilotare il passaggio da SIMD a MultiSIMD (e viceversa) tramite istruzioni ad hoc, modificando a runtime la struttura dell'insieme di SISMA-C. In tal modo dati di input che devono subire elaborazioni successive possono fluire da un SISMA-C all'altro senza passare tramite i dispositivi di I/O (modalità SIMD), per essere poi elaborati parallelamente su SISMA Computer indipendenti (modalità MultiSIMD) (fig. 3).

Ogni SISMA - C dovrà assolvere a due compiti principali:
- effettuare la *COMPUTAZIONE SISTOLICA,* cioè eseguire le istruzioni tipiche del GAPP e quelle di configurazione peculiari di SISMA.
- *FORNIRE LE ISTRUZIONI A SISMA* e *CONTROLLARNE IL FLUSSO,* in particolare sincronizzare il controllo dei diversi SISMA - C nel passaggio SIMD/MultiSIMD e gestire le operazioni di O.

Tali funzionalità identificano i blocchi logici rispettivamente denominati SISMA e CONTROLLER (fig.l).

3.1. Funzionalità SISMA

SISMA é definito da 3 entità logiche (fig.l):
- il GAPP ARRAY (GA) in cui si svolge la computazione sistolica vera e propria. Può essere riconfigurato via SW a runtime nelle seguenti tipologie: toro, cilindro orizzontale o verticale, stringa unica orizzontale, doppia stringa orizzontale, aperto nelle 4 direzioni.
- il CORNER TURN (CT), il cui compito é di trasformare i dati in input per GA dal formato bit-parallelo/word-seriale a quello caratteristico del GAPP word-parallelo/bit-seriale.
- l'INTERFACE, che si interpone tra Controller e GA-CT, instradando le istruzioni provenienti dal primo verso le rispettive unità di esecuzione.

3.2. Funzionalità Controller

Durante una sessione di lavoro sulla workstation sistolica, accadrà che un processo attivo sull'host, ad un certo punto, richiederà l'intervento del coprocessore SISMA-C, il quale, in generale, effettuerà le sue elaborazioni parallelamente all'attività dell'host. Nella situazione più elementare, una subroutine scritta in SL prevederà una fase di input di dati, una loro successiva computazione ed infine un output dei risultati. In tal caso il Controller dovrà semplicemente prelevare un'istruzione alla volta dal codice SL ed inviarla a SISMA, indirizzandola correttamente al solo CT, al solo GA o ad entrambi se essa é composta. La scelta dell'istruzione successiva é determinata dalle istruzioni di controllo di flusso del programma (espressioni condizionali, cicli, ...) eseguite direttamente dal Controller; in particolare, oltre alle consuete operazioni su variabili interne (es: incremento di indici), andranno gestite anche le diramazioni del codice determinate dal valore del Global Output, il quale è l'unico dato globale sullo stato della computazione che il chip GAPP, e quindi SISMA, fornisce. Nelle fasi di input, come in quelle di output, il Controller deve, infine, sincronizzarsi con le periferiche di I/O sospendendo eventualmente l'esecuzione finché sia SISMA sia il dispositivo di I/O coinvolto non siano pronti a scambiarsi i dati. Il quadro così prospettato sottoutilizza le potenzialità di SISMA-C, limitando le situazioni di parallelismo a quelle esistenti tra host e coprocessore, tra GA e CT, oltre a quelle già viste nel chip di base.
E' possibile fare di meglio.
Normalmente la workstation comprende più unità SISMA-C ed esistono, perciò, diversi Controller che devono cooperare tra loro. Nel caso SIMD é logicamente attivo un solo Controller, che si comporta da master nei confronti degli altri, mentre in MultiSIMD tutti i Controller sono attivi; il passaggio SIMD/MultiSIMD via software necessita, allora, di una sincronizzazione tra Controller. Manca ancora un ultimo livello di parallelismo, non meno significativo dei precedenti in quanto limita l'incidenza del collo di bottiglia dovuto all'I/O.Generalmente gli algoritmi di input/output si SISMA sono sovrapponibili con quelli di computazione, ma demandare questo compito al programmatore SL non consentirebbe la sovrapposizione nei casi in cui il numero di istruzioni di almeno uno dei due algoritmi non sia noto al compilatore; si é scelto, quindi, di lasciare anche questa funzione al Controller, il quale si occuperà di comporre le istruzioni dei due algoritmi per formare l'istruzione eseguibile di SISMA [3].
La descrizione funzionale del Controller compare rias-

sunta nella fig.4.

4. LINGUAGGIO E TOOLS DI SVILUPPO SW

I requisiti principali cui deve soddisfare l'ambiente di sviluppo software della workstation sistolica prevedono l'esistenza di un linguaggio di programmazione (SL o SISMA Language), di un debugger e di un'interfaccia SW che consenta all'utente di monitorare runtime lo stato della computazione su SISMA.
SL dovrà consentire di sfruttare appieno le potenzialità di SISMA-C, comprendendo istruzioni per il chip al più basso livello di granularità (GAPP instruction set), istruzioni per la riconfigurabilità software di SISMA-C e costrutti per gestire la parallelizzazione tra operazioni di I/O e operazioni di computazione; SL dovrà anche avere le caratteristiche tipiche di un linguaggio ad alto livello.
Motivi di economia e priorità di sviluppo del progetto globale hanno spinto, per ora, a tralasciare la realizzazione del compilatore per SL e ad utilizzare un linguaggio già esistente, supportato da opportune funzioni di libreria che gestiscono l'invio delle istruzioni a SISMA; ciò ha permesso, come già accennato, di non realizzare il Controller in quanto le sue funzionalità vengono assolte dal runtime del linguaggio stesso.
Così, la programmazione di SISMA-C é attualmente consentita da una libreria di funzioni C, linguaggio scelto per la sua diffusione sui sistemi di calcolo tradizionali e per il fatto che SISMA-C é nato come coprocessore da inserire in tali ambienti; ciò, tra l'altro, assicura una naturale integrazione tra codice per l'host e codice per SISMA. Queste funzioni C-SL corrispondono ad un sottoinsieme delle specifiche date per SISMA Language; in particolare alcune di esse identificano univocamente l'instruction set del chip di base e si rivolgono o a CT o a GA o ad entrambi, altre corrispondono alle istruzioni di configurazione del GAPP Array, altre ancora consentono di testare il valore del GO e di eseguire l'I/O dei dati su CT.
I costrutti per gestire la parallelizzazione computazione-I/O e le istruzioni di sincronismo per il passaggio SIMD/MultiSIMD, se realizzate in SW, sono troppo inefficienti per essere utili in programmi applicativi; per questa ragione il prototipo ne é sprovvisto e la loro implementazione é rimandata alla realizzazione del Controller HW.Oltre che di un linguaggio di programmazione, l'utente necessita di un ambiente adeguato per il debugging di programmi sistolici; a tale scopo ha a disposizione un'interfaccia SW, sviluppata per interagire con SISMA, e può utilizzare un qualunque debugger C per il codice SL. Poiché durante la fase di messa a punto di un codice sistolico é opportuno conoscere a runtime lo stato della computazione in punti prestabiliti, l'interfaccia SW consente di accedere, usando nomi di variabili simboliche al contenuto della RAM e dei registri dei chip e di controllare il procedere dell'esecuzione, sospendendola nei punti desiderati.

Per completare l'ambiente di sviluppo di programrni sistolici, rendendolo indipendente dalla presenza dell' hardware SISMA, é stato realizzato un simulatore SW ch corrisponde completamente alle caratteristiche di SISMA stesso. La trasparenza a livello utente é, comunque, garantita dal fatto che la libreria di funzioni C-SL invia le istruzioni a SISMA, in caso di esistenza dell' HW, mentre esegue il codice che costituisce il simulatore, altrimenti. Ovviamente, anche l'interfaccia SW sopra descritta é utilizzabile sia con la piastra che con il simulatore.

5. DESCRIZIONE STRUTTURALE SISMA

Il prototipo della workstation sistolica in corso di realizzazione si scompone, al primo livello di astrazione, in due blocchi: SISMA e HOST computer (l'host, come già detto, realizza via SW anche le funzionalità del controller) (fig. 5).

5.1. Host Computer

L' HOST é un calcolatore "tradizionale" composto da:
- una scheda equipaggiata da una CPU Motorola 68020 a 20Mhz conforme allo standard VME bus e caratterizzata da alte prestazioni. La scheda può essere usata in un sistema a singolo processore, oppure come processor element in un sistema a multiprocessore. Il calcolatore é supportato "on board" da 1 Mb di RAM statica accessibile senza cicli di wait.
- una scheda grafica per l'aquisizione di immagini da videocamera. L'host dispone del sistema operativo OS-9/68020.

5.2. SISMA

SISMA é il coprocessore sistolico della workstation ed

é strutturalmente composto da tre blocchi (fig. 6):
- *GAPP ARRAY*
- *CORNER TURN*
- *INTERFACE*

5.2.1. GAPP Array

GA é un array sistolico programmabile e configurabile composto di 2304 PE a singolo bit sincronizzati da un unico segnale di clock. E' realizzato con 32 chip GAPP NCR45CG72 topologicamente disposti in un array 48x48 PE .
GA costituisce l'anima computazionale del sistema di elaborazione sistolica ed in esso si realizzano le caratteristiche sottolineate di modularità, regolarità e configurabilità del sistema.
La configurazione dell'array sistolico può essere selezionata e modificata dinamicamente attraverso l'opportuna gestione dei blocchi di configurazione presenti in GA.
I blocchi di configurazione sono pilotati direttamente dalle istruzioni di configurazione e gestiscono le connessioni delle frontiere N, S, E, W dell'array, selezionando il cammino dei dati. Il blocco GA é stato progettato in modo tale da prevedere espansioni sia verticali che orizzontali dell'array e affinché la complessità delle connessioni tra le schede fosse la minore possibile.

5.2.2. Corner Turn

CT é un array sistolico programmabile e configurabile composto da 1152 PE a singolo bit. E' realizzato con 16 chip GAPP topologicamente disposti in array 24x48 PE ed é anch'esso sincronizzato da un unico segnale di clock. Le funzionalità di formattamento dei dati in I/O per il GA sono state implementate in questo blocco. A tal proposito il grado di libertà dell'array CT é stato ridotto, cioé un sottoinsieme di istruzioni GAPP sono state implementate HW al fine di specializzare il CT al solo formattamento dei dati.

5.2.3. Interface

Interface é una scheda di interfaccia progettata ad hoc per la comunicazione tra il blocco SISMA e l'HOST che svolge anche le funzioni di controller. L'HOST "vede" SISMA come RAM, nel senso che un sottoinsieme dello spazio di indirizzamento della CPU é stato dedicato al colloquio tra i due blocchi.
Strutturalmente INTERFACE si compone di:
- un blocco di decodifica indirizzi, tra i quali é possibile evidenziare: ind. istruzione GA, ind. istruzione CT, ind. configurazione GA, ind. configurazione CT.
- un blocco di generazione segnali. In particolare in questo blocco é gestita la generazione dei segnali di acknoledgement per la CPU, che lavora in modo asincrono, ed i segnali di clock per GA e CT .
Nel blocco INTERFACE é inoltre gestita la selezione della modalità di funzionamento SIMD o MultiSIMD del sistema.

Il prototipo sperimentale della workstation sistolica disporrà di una coppia di SISMA ognuno composto da due unità GA, un CT ed una INTERFACE, per comporre un array di 96x96 PE.

5.3. Configurabilità dell'architettura

In una architettura SISMA i blocchi GA e CT vengono configurati attraverso l'invio di opportune istruzioni di configurazione. Le configurazioni supportate da GA sono di tre tipi:

- *ORIZZONTALE*
- *VERTICALE*
- *STRINGA*

A partire da questi tre tipi di configurazioni é possibile effettuare delle composizioni tra le stesse, ad esempio: orizzontale/verticale e stringa/verticale, ciò attribuisce al sistema un elevato grado di configurabilità. Per quanto riguarda la configurabilità orizzontale del sistema é possibile realizzare degli array bidimensionali chiusi oppure aperti con ingresso selettivo di 0/1 durante le operazioni di shift (E, W) dei dati. In modo analogo é possibile gestire le frontiere verticali (N,S) dell'array [3]. Per le configurazioni di tipo stringa é da evidenziare la possibilità di connettere l'array sistolico in un unica spirale, a sviluppo orizzontale, chiusa, o, con eventuale ingresso di 0/1 dalle estremità [3]. In tal caso, se il sistema é composto da una unità SISMA base 48x48, si otterrà una stringa di 2304 PE. L'array può configurarsi anche in due stringhe parallele, chiuse su loro stesse oppure aperte con ingresso selettivo di 0/1.
Il blocco CT é configurabile solo orizzontalmente per

comporre array di dimensioni maggiori, e si predispone al corretto fommattamento dei dati in I/O solo dopo aver ricevuto l'istruzione di configurazione ad esso diretto.

Il numero ed i tipi di configurazione supportate dalla unità base SISMA 48x48 PE sono ottenibili in una qualsiasi espansione verticale o orizzontale dell' array di PE.

6. CONCLUSIONI

Il progetto descritto nasce da una collaborazione tra D.S.I. e Me.S.A. ed ha lo scopo di verificare le potenzialità reali di applicazione di un elaboratore sistolico con le caratteristiche di SISMA. Lo sviluppo del prototipo e del software di base presentati é proseguito in parallelo alla realizzazione di algoritmi sistolici sulla workstation provvista del simulatore SISMA. Gli algoritmi sviluppati sono stati orientati, da un lato, a problemi sistolici tradizionali - e. g. elaborazione delle immagini - dall'altro a problemi non standard per questo tipo di processori, quali il pattem matching e l'unificazione nell'ambito dell'elaborazione simbolica. Inoltre l'uso del prototipo consentirà di valutare l'aumento di prestazioni ottenibile realizzando un Controller specifico e di definire le modifiche dell'architettura, e in particolare del chip di base, adeguate per rendere la workstation sistolica più flessibile e potente.

7. RINGRAZIAMENTI

Ringraziamo il dott. Dario Mella e il dott. Fabio Bemo le cui tesi di laurea hanno dato inizio al progetto SISMA. Ringraziamo il prof. Gianni Degli Antoni la cui costante supervisione del progetto ha consentito di ottenere i risultati presentati, e il prof. Angelo Beltramini per i suggerimenti e i mezzi forniti. Ringraziamo inoltre il dott. Mauro Fiorentini per la realizzazione del simulatore e i consigli per l'implementazione dell'ambiente di sviluppo SW e il dott. Tommaso Majo per la collaborazione in fase di progettazione dell'HW di SISMA.

8. BIBLIOGRAFIA

[l] H.T. Kung, WHY SYSTOLIC ARCHITECTURES?, Computer Magazine 15(1):37-46, gennaio 1982

[2] S. Y. Kung, VLSI ARRAY PROCESSOR, Prentice Hall, 1988

[3] D.S.I., PROGETTO S.I.S.M.A. - SYSTOLIC INTEGRATED SOFTWARE MODIFIABLE ARCHITECTURE, Workshop: "Processori Sistolici: applicazioni e prospettive", febbraio 1989

[4] NCR "GAPP application note" [1- 5], 1988

AVVENIMENTI

HYPERTEXT '89

Paolo Fezzi, Roberta Macchi
Hypertext User Group
Dipartimento di Scienze dell'Informazione
Università di Milano

Il secondo atteso incontro sugli ipertesti, sponsorizzato da ACM, Apple, IBM, Knowledge Systems, Texas Instruments, si è tenuto a Pittsburgh dal 5 al 8 novembre 1989 con un programma molto intenso: 11 *tutorial*, 9 *paper session* in concomitanza con altrettante *panel session*, 15 dimostrazioni e 4 incontri *Birds of a feather.*

Prospettive di sviluppo

È questo l'argomento su cui N. Meyrowitz ha tenuto il suo discorso inaugurale. Confrontando le fantascientifiche caratteristiche del celebre Memex, vagheggiate da Bush nel lontano 1945, con lo stato attuale dell'arte, Meyrowitz ha delineato quelle che saranno le sfide del futuro nello sviluppo di tools ipertestuali, prendendo spunto dalle caratteristiche attuali e dalle possibilità evolutive di Intermedia, sistema alla cui realizzazione ha attivamente collaborato.

Egli ha innanzitutto sottolineato l'utilità per l'utente di evidenziare le parti di documento da lui considerate rilevanti e di renderle permanenti (ovvero disponibili ogniqualvolta viene aperto il file). Una selezione permanente di questo tipo è da Meyrowitz detta *àncora* per la sua funzione di punto di riferimento fisso, non soltanto per l'attenzione del lettore ma anche per eventuali riferimenti ipertestuali. Infatti un'*àncora* può essere sorgente o destinazione di link. Le *àncore*, che possono avere nomi, attributi/valori, parole-chiave, devono essere indipendenti dall'applicazione: possono comprendere, cioè, indifferentemente sequenze di testo, parti di immagini, celle di spreadsheet, ecc.

Per rendere la struttura specificamente ipertestuale dei documenti indipendente dal loro contenuto, con la possibilità di manipolare la struttura ipertestuale senza intaccare il contenuto informativo e di estendere l'applicabilità dei link, Meyrowitz pensa che sia indispensabile memorizzare i link su file distinti rispetto a quelli contenenti i documenti. Inoltre è opportuno, a suo parere, che i link siano bi-direzionali: deve essere sempre possibile da un documento accedere ai documenti che si riferiscono ad esso.

Meyrowitz ritiene importante l'uso dei link non solo per il passaggio da un documento all'altro (link navigazionali) ma anche come canale per il trasferimento di dati: in tal caso egli parla di "warm links".

Per facilitare la gestione del testo e dei link in direzione di una maggiore automazione, il relatore suggerisce la possibilità di definire delle *àncore* come istanze di una *àncora- master*: in tal modo tutte le modifiche sul testo e sui link dell'*àncora-master* vengono riprodotte

automaticamente nelle *àncore-istanze*. Il collegamento fra *àncora-master* e le *àncore- istanze*, che permette la loro completa sincronizzazione, è detta da Meyrowitz "hot link".

Sempre in direzione di una automatizzazione nella creazione dei link, è stata sottolineata l'opportunità di fornire al sistema procedure di *pattern matching* sull'origine/destinazione dei link, in modo tale che ogniqualvolta nel testo compare una certa parola o una certa frase, essa venga automaticamente collegata ad una stessa destinazione. Tali procedure vanno comunque affiancate a strumenti interattivi di controllo, che permettano all'utente di gestire l'automatizzazione in maniera intelligente.

Nella visione di Meyrowitz, i link dovranno in futuro tener conto della dimensione temporale, nella misura in cui avremo a che fare sempre più con oggetti dinamici, come animazioni, video-clips, voce e musica. In altre parole, insieme ai link, dovranno essere rappresentati gli intervalli di tempo della loro destinazione, ad esempio 30 frame di animazione o 8 battute musicali.

In documenti di grandi dimensioni è opportuno introdurre tecniche di contestualizzazione, in modo tale che vengano attivati solo i link pertinenti agli interessi dell'utente. Intermedia fornisce già la possibilità di definire di volta in volta un link come appartenente ad un determinato contesto, detto "web": all'inizio di una sessione di lavoro vengono attivati uno o più "web"; di conseguenza, aprendo un certo documento, saranno resi visibili e disponibili soltanto i link relativi ai "web" attivi in quel momento. Meyrowitz propone, inoltre, come tecnica alternativa di contestualizzazione quella dei filtri. Mediante l'assegnazione di attributi a documenti, ancore e link, diventa possibile rendere accessibili, "filtrandoli", solo quei documenti, quelle ancore e quei link che possiedono certi attributi, relativi, per esempio, alla data di creazione o all'argomento.

Per quanto concerne la navigazione degli ipertesti (*browsing*) Meyrowitz ha auspicato il miglioramento delle mappe, come strumento di orientamento; ha suggerito di potenziare le funzionalità di *hystory*, fornendo procedure per l'apertura, la chiusura e l'attivazione di percorsi ritenuti interessanti dall'utente; ha evidenziato, inoltre, l'opportunità di trattare i *path*, cioè le sequenze di nodi attraversabili, come oggetti unitari che possono essere editati mediante le usuali procedure di *cut/copy/paste*.

Le funzionalità ipertestuali si integreranno sempre più con sistemi di *full-text indexing* e *retrieval*, al punto da diventare, a detta di Meyrowitz, parti di un unico sistema. Diventeranno, inoltre, disponibili *utilities* come la consultazione on-line di dizionari, thesauri, correttori grammaticali e sintattici.

Sembra, quindi, che l'ipertesto, da applicazione particolare, sia destinato a diventare l'interfaccia fra diverse applicazioni, disponibile come funzionalità di sistema; nella comunicazione fra le varie applicazioni, ha concluso il relatore, si passerà così dalla *clipboard* alla *"linkboard"*.

Convertibilità fra sistemi ipertestuali

Le piattaforme ipertestuali nei laboratori di ricerca stanno diventando sempre più numerose. Benchè questa rapida proliferazione possa ad un certo punto arrivare a determinare una situazione stabile, è quasi certo che nessuna delle parti in gioco dominerà il mondo degli ipertesti in modo da divenire lo standard de facto. La consapevolezza di questa situazione ha fatto nascere l'esigenza di una standardizzazione formale dei sistemi ipertesto. Un'indagine di mercato ha, d'altronde, rilevato come questa sia un'esigenza assai diffusa.

Alcuni fra i personaggi più rappresentativi nel campo degli intertesti, hanno già formato un gruppo di lavoro (il Dexter Group) che sta lavorando all'idea di comunicazione fra sistemi diversi. Del gruppo fanno parte: F. Halask (Notecards), R. Akcsyn (KMS), T. Oren e J. Bornstein (Apple) e V. Riley (Intermedia). Questi ricercatori hanno già dato delle indicazioni su di un possibile formato comune (il Dexter Hypertext Interchange Format - DHIF) sollecitando anche contributi da parte dei presenti, in modo che l'Interchange Format includa in sè tutte le possibili definizioni di ipertesto.

Benchè l'architettura che si sta elaborando si sforzi di riflettere le proprietà dei diversi sistemi, essa non potrà comunque tenere conto di tutte le loro caratteristiche, data la diversa natura dei paradigmi alla base dei tools ipertestuali.

Ad esempio, KMS ed HyperCard utilizzano le card per rappresentare le informazioni, mentre Intermedia e Notecards preferiscono un modello più fluido di rappresentazione. Importare quindi un documento di Notecards in HyperCard necessita la ridefinizione di un paradigma che indicherà come le informazioni di Notecard verranno visualizzate nell'ambiente ipertestuale "straniero".

Nel Dexter Hypertext Interchange Format tutte le informazioni relative ad una applicazione vengono tradotte in un file DHIF, in modo tale che se un documento viene esportato, modificato e reimportato in un sistema, la versione corrente del file DHIF rifletta le modifiche che sono state apportate.

Uno dei problemi maggiori del DHIF riguarda l'interpretazione del contenuto procedurale dei link. In sostanza, esistono sistemi che consentono di associare ad un link un programma che controlla la destinazione del link stesso, mentre altri sistemi non hanno questa funzionalità. Una soluzione che è stata proposta intende fare in modo che l'interpretazione del contenuto procedurale possa preservare almeno una delle possibili destinazioni del link. I programmi associati ai link rappresentano ancora un problema quando fanno divenire importante l'ordine in cui i link vengono attraversati.

Attualmente il DHIF è un formato ASCII e ha una struttura gerarchica che lo rende simile allo Standard Generalized Markup Language (SGML). Le implementazioni parziali per ora sono soltanto due: quella di F. Halask che traduce Notecards in DHIF e quella di J. Bornstein che realizza la traduzione in HyperCard dei file DHIF generati da Notecards. E inoltre in corso di realizzazione, da parte di F. Halask, la descrizione matematica del modello Dexter.

Metodologie per la costruzione e la navigazione di ipertesti

Si avverte, ormai, l'esigenza di definire delle metodologie per la strutturazione di ipertesti, per facilitare l'autore a costruirli e l'utilizzatore a consultarli senza perdersi. Tali metodologie hanno interessanti implicazioni a livello di implementazione: una volta individuate le strutture tipiche può diventare conveniente automatizzarne la creazione e utilizzarle come base per ulteriori elaborazioni.

TOPOLOGIE DI IPERTESTI

H. Van Dyke Parunak dell'Industrial Technology Institute di Ann Arbor MI, ha cercato, nel suo intervento, di definire le più ricorrenti topologie di ipertesto. A suo parere esse si possono ricondurre a quattro tipi fondamentali: la struttura lineare, quella gerarchica (ad albero binario), quella ipercubica e quella a lattice. Ha poi fatto una interessante indagine su come le diverse strategie di navigazione si possano applicare a ciascuna topologia.
Se, da un lato, le diverse possibili strutture di un ipertesto possono essere rappresentate da metafore di tipo spaziale (da cui il nome di topologie), dall'altro, le strategie di navigazione ricalcano, secondo Parunak, le modalità di orientamento dell'uomo nello spazio fisico. L'autore riconduce, infatti, le strategie di orientamento a cinque categorie fondamentali: l'orientamento mediante identificazione, nella quale si confronta il nome del luogo cercato con nomi dei luoghi raggiungibili; il path o cammino, cioè una descrizione procedurale dei passi necessari per arrivare al luogo desiderato; la distanza, ovvero la misura della lontananza dal luogo di destinazione ed infine l'indirizzo, che consente nel fornire le coordinate tridimensionali del punto di arrivo.

INTERFACCE GRAFICHE COME RAPPRESENTAZIONE DI STRUTTURE IPERTESTUALI

Un tema che è stato toccato in molti interventi è quello della rappresentazione della struttura degli ipertesti mediante interfacce grafiche, utili come ausilio alla navigazione in ipertesti complessi. Gli strumenti più citati a riguardo sono le cosiddette mappe, grafi più o meno sofisticati in grado di visualizzare i nodi e i link, che rivelano tuttavia il loro limite nel caso di ipertesti in cui il numero dei link è di molto superiore al numero dei nodi.

Una metafora alternativa è quella proposta da M. Traves del Media Lab (MIT), il quale concepisce le basi di conoscenza di grandi dimensioni come musei, in cui ogni unità informativa (frame) costituisce una stanza. Dal momento che ogni stanza può contenere oggetti,

ogni frame può essere, rispetto agli altri, sia una stanza che un oggetto. Sulla base di questa metafora, M. Treves ha costruito, nell'ambito del progetto Cyc, un'interfaccia per basi di conoscenza, da lui battezzata MUE (Museum Unit Editor) in cui la struttura della base di conoscenza è rappresentata da un insieme di scatole nidificate (con un video ad alta risoluzione si possono visualizzare fino a sei livelli di *nesting*). L'inclusione di una scatola in un'altra rappresenta una relazione concettuale tra due unità informative; il colore delle scatole il tipo di relazione, la sua intensità la posizione nella gerarchia.

Negli esempi presentati dall'autore venivano trattate soltanto relazioni gerarchiche del tipo classe/sottoclasse, tutto/parte. Non è chiaro perciò se ed in che modo vengano rappresentati altri tipi di relazioni. Qualora altri tipi di relazione non siano contemplati, il limitarsi a relazioni di tipo gerarchico sarebbe estremamente riduttivo; nel caso, invece, che con l'inclusione spaziale si voglia rappresentare qualsiasi tipo di relazione, la metafora utilizzata, di una semplicità accattivante nel caso di strutture gerarchiche, sembra, invece, assai meno adeguata ed intuitiva per relazioni concettuali non gerarchiche.

IPERTESTI E RETI DI PETRI

R. Furuta e P.D. Stotts hanno presentato αTrellis, un sistema per la creazione e la navigazione di ipertesti, basato sulle reti di Petri, formalismo che permette non solo una rappresentazione grafica delle connessioni ipertestuali ma anche di esprimere vincoli e condizioni per l'attivazione dei link. Le reti di Petri, infatti, costituiscono un potente linguaggio di rappresentazione dei processi, riconducendoli alle nozioni base di eventi e condizioni per il verificarsi di questi eventi; questo linguaggio può essere tradotto sia in forma grafica intuitiva che in rigorose regole formali. Può perciò, da un lato essere usato per la costruzione di interfaccie grafiche, dall'altro, considerando l'attivazione di un link come evento, permette di esprimere i vincoli ai quali tale attivazione è subordinata. Ad esempio, la visualizzazione di un link può essere subordinata, in maniera specificata dall'autore, alla visitazione di uno o più nodi precedenti.
αTrellis fornisce un linguaggio di programmazione, detto Alphα, per strutturare gli ipertesti in base alle reti di Petri.
Ci sembra, tuttavia, che le reti di Petri, come tutti i formalismi, da sole siano insufficienti a rappresentare la ricchezza e la complessità delle relazioni intertestuali: in certi casi, perciò, l'applicazione delle reti di Petri può costituire una forzatura.

PROCEDURE PER IL TRATTAMENTO DEI PATH

Da più parti si suggerisce l'opportunità di trattare i *path* (o cammini) non come semplici liste di link ma come oggetti unitari. In tal modo sarebbe possibile costruire delle procedure per il loro trattamento, allo scopo di facilitare la costruzione e la navigazione di ipertesti.

E stato questo, in particolare, l'argomento dell'intervento di T. Polle Zellweger, che allo Xerox Parc sta studiando in primo luogo procedure di editing tali da permettere la sostituzione, la cancellazione e l'aggiunta di nodi intermedi nell'ambito di un path senza compromettere l'integrità del path stesso; in secondo luogo procedure di ricerca di path rilevanti rispetto agli obiettivi dell'utente.

APPROFONDIMENTO DELLA NOZIONE DI LINK

Quali significati e quali funzioni possono assumere i link ipertestuali? Cercando di rispondere a questa domanda, si tende a delineare una vera e propria semantica dei link. Tuttavia nessuna tassonomia potrà mai essere esaustiva. Un obiettivo del genere rischia, quindi, di mettere capo ad un compito senza fine, come infinite sono le sfumature e i linguaggi, o di scadere nella mera accademia.

Una semantica dei link può, peraltro, darci una indicazione di massima, sia pure provvisoria, di tutti i campi in cui attualmente si dispiegano le potenzialità degli ipertesti e porre le basi per una tipizzazione automatica dei link.

Un contributo onesto, ma forse prematuro, in questa direzione viene da S.J. DeRose, un linguista che ha partecipato alla realizzazione, mediante Guide e CDWord, della Bibbia ipertestuale al seminario teologico di Dallas. La terminologia usata in questo primo abbozzo di

tassonomia ha tuttavia il limite di essere artificiosa ed esoterica, al punto che si dubita possa diventare di uso comune. Il pregio di questo tentativo consiste, invece, nel segnalare, sulla base dell'esperienza diretta, alcune direzioni di sviluppo non ancora esplorate a fondo, come i cosiddetti "link isomorfici", link che mettono in relazione parti omologhe di testi, aventi cioè la stessa struttura, come, ad esempio, diverse traduzioni di un testo.

STRUTTURAZIONE LOGICA DEGLI IPERTESTI

Si sente l'esigenza di fare degli ipertesti strumenti per evidenziare le strutture logiche del discorso. Alla base vi è l'idea che un'ipertesto può essere visto come una rete semantica e che, di conseguenza, è possibile tipizzare i link in maniera tale che riflettano i diversi tipi di relazioni concettuali. In questo modo si perseguono due scopi fondamentali: il primo obiettivo è facilitare la comunicazione, favorendo, da parte del lettore, la sua comprensione della logica del testo, e, da parte dello scrittore, il processo di traduzione delle idee in forma scritta. Il secondo fine è predisporre il testo all'analisi automatica dei suoi aspetti argomentativi e retorici.

La prima realizzazione, in questo campo, si può considerare il sistema gIbis, sviluppato da J. Conklin su workstation, di cui sotto diamo una breve descrizione.

Un più ambizioso progetto in questo ambito è il sistema SEPIA (Structured Elicitation and Processing of Ideas for Authoring) presentato da N.A. Streitz della Gesellschaft fuer Mathematik und Datenverarbeitung mbH di Darmstadt (RFT). La scrittura è, secondo Streitz, un *work- in-progress* che va dal caos all'ordine seguendo fasi successive di strutturazione. Sepia fornisce allo scrittore un ausilio alla sua attività di stesura ed organizzazione del testo, predisponendo quattro distinti spazi di lavoro:

- lo spazio di ideazione, una sorta di *outline processor* che consente la suddivisione in temi e sottotemi;
- lo spazio di contenuto, che permette di raccogliere ed associare ideee in maniera ancora informale, collegandole, eventualmente, a citazioni di documenti già esistenti;
- lo spazio argomentativo, che ci è sembrata la parte più originale del progetto: ogni argomentazione a favore o contro una certa tesi è suddivisa in affermazione, premessa, giustificazione generale della premessa ed eventuale fondamento teorico ancora più generico;
- lo spazio retorico, che consente di realizzare l'organizzazione complessiva del discorso, predisponendone la trasposizione su carta.

DAL TESTO ALL'IPERTESTO: IL PROBLEMA DELLA CONVERSIONE AUTOMATICA

Una tematica assai dibattuta e destinata a diventare sempre più attuale, con la diffusione degli ipertesti e la crescita delle loro dimensioni, è quella della conversione automatica dei testi in ipertesti. Supponendo infatti di avere a disposizione documenti di grandi dimensioni, già in forma elettronica, si pone il problema di automatizzare, sia pure in maniera parziale, il passaggio della struttura sequenziale tradizionale a quella ipertestuale.

Lo studio di questo problema è ancora a uno stadio embrionale: in questo ambito è stato fatto ancora molto poco e si sono poste solo le premesse.

Il punto di partenza è dato dalla possibilità di individuare nei testi strutture tipiche, che sono implicitamente ipertestuali, in quanto costituiscono altrettante modalità di rimando da un testo all'altro. La tipicità di queste strutture renderebbe appunto possibile automatizzarne il riconoscimento e la conversione in rimandi ipertestuali. Tra queste strutture le più ricorrenti sono le note a piè pagina, i rimandi bibliografici, la strutturazione gerarchica in parti, capitoli, paragrafi e sottoparagrafi e, per quanto concerne manuali ed enciclopedie, i rimandi incrociati da una voce all'altra.

Se è evidente la possibilità di tradurre tali strutture in riferimenti ipertestuali (per esempio, nel caso della struttura gerarchica, collegando ai titoli di ciascun capitolo i titoli dei suoi paragrafi e a quest'ultimi, nel caso che non abbiano ulteriori suddivisioni, il loro contenuto), poco è stato detto sulle modalità di attuazione del loro riconoscimento automatico. Non è stato, inoltre, chiarito se convenga convertire il testo in un formato di mark-up intermedio, il più possibile neutrale, o direttamente nel formato di uno specifico tool ipertestuale in commercio.

Se dunque la discussione rimane ancora a un livello teo-

rico, si è anche parlato di una prima realizzazione concreta in questo campo: si tratta del progetto HyperBook, sviluppato su workstation Sun e probabilmente disponibile come prodotto già l'anno venturo, nei confronti del quale c'è una certa attesa, ma che è ancora circondato da un certo riserbo.

Ipertesti e sistemi esperti

Il tema della relazione fra ipertesti e sistemi esperti è stato affrontato in varie sedi e si prefigura più come un rapporto di complementarietà ed integrazione che di competizione. Da un lato, infatti, gli ipertesti, grazie alla loro flessibilità ed apertura, promettono di essere un interessante strumento per estendere la documentazione on-line nei sistemi esperti e di ovviare, inoltre, ad uno dei loro punti deboli, quello della acquisizione della conoscenza; d'altro lato i sistemi esperti possono fornire quelle sofisticate euristiche di ricerca necessarie per la navigazione di ipertesti complessi.

D'altronde, è proprio la diffusa esigenza di una strutturazione degli ipertesti a spingere verso la loro assimilazione ai sistemi esperti, al punto che M. Bernstein, della Eastgate Systems Inc., ha ironicamente osservato che, se un tempo credeva di sapere che cosa distingueva un ipertesto da un sistema esperto, al momento attuale gli sarebbe difficile indicarlo.

Se la tendenza verso l'integrazione è evidente, manca tuttavia un paradigma riconosciuto per realizzarla: è significativo, a riguardo, che prototipi come Janus o IDE, che vanno in questa direzione, abbiano suscitato, in sede di presentazione, feroci polemiche e ben pochi consensi.

Information retrieval ed indicizzazione

Un tema assai dibattuto è stato quello del rapporto tra tecniche di information retrieval e ipertesti. I contributi più rilevanti, ad opera di B. Croft, dell'Università del Massachusetts e di M. Frisse, dell'Università di Washington, riguardano l'applicazione di tecniche probabilistiche (logica bayesiana) a documenti ipertestuali.

B. Croft ha proposto di applicare un modello probabilistico, associando a ciascun nodo di un ipertesto un indice di rilevanza rispetto al concetto che costituisce l'obiettivo della ricerca.
Dal canto suo, M. Frisse sta approntando strumenti interattivi per far dipendere il valore degli indici di rilevanza dall'effettivo interesse dell'utilizzatore, offrendo a quest'ultimo la possibilità di segnalare al sistema il suo interesse o non interesse rispetto al contenuto del nodo consultato.

Nel corso di un seminario è stata inoltre rilevata l'analogia fra l'indicizzazione e la creazione dei link ipertestuali: in entrambi i casi il problema principale consiste nel decidere che cosa è rilevante rispetto agli interessi dell'utente, dipendendo il criterio di rilevanza dal background culturale e dagli obiettivi dell'utente stesso.

Ipertesti e didattica

La discussione voleva essere un momento di discussione sull'applicazione alla didattica non solo degli ipertesti ma delle tecnologie della informazione in generale.

Le esperienze raccolte durante il dibattito hanno evidenziato che sull'uso delle tecnologie didattiche esiste una spaccatura fra docenti di discipline scientifiche ed insegnanti di materie letterarie, i quali mantengono ancora una certa diffidenza nei confronti dell'informatica.

I primi traggono dagli ipertesti ausilii importanti per l'insegnamento: "Gli studenti sono felici di utilizzare questi strumenti e la loro preparazione è più precisa e profonda". I docenti di discipline umanistiche, invece, hanno aspramente criticato l'uso indiscriminato del computer nelle loro materie, sostenendo che l'elaboratore elettronico non è realmente uno strumento interattivo, in quanto la varietà delle risposte che fornisce è sempre troppo povera in rapporto alla varietà degli interessi dell'utilizzatore. Inoltre, sostengono, l'uso del computer è particolarmente indicato nel lavoro di equipe; ma, se lo studio collettivo può essere applicato con ottimi risultati alle materie scientifiche, non è detto che gli stessi effetti si abbiano nelle materie umanistiche, nelle quali l'apprendimento ha caratteristiche più spiccatamente individuali.

Il problema aperto è: riusciranno gli ipertesti, che molti vedono come una via verso una maggiore flessibilità

nella comunicazione uomo-macchina, ad aprire una breccia nella diffidenza degli umanisti verso i computer?

Interactive Fiction

La comparsa degli ipertesti spinge ad un confronto fra letteratura e *computer science*. Il dibattito ha evidenziato due punti di vista differenti: quello di S. Moulthrop, della Yale University, che offre una prospettiva legata più alla sociologia che alla critica letteraria, e quello di M. Joyce che, nella sua disanima, si attiene maggiormente alle influenze che le nuove tecnologie della informazione possono avere sul testo scritto e sul pensiero individuale.

Gli ipertesti, secondo S. Moulthrop, sono l'implementazione effettiva di un movimento culturale che coincide con l'ultima fase della modernità. Questo movimento rifiuta le gerarchie del linguaggio, lineari e deduttive, a favore di sistemi di discorso che riconoscano una pluralità di significati. Da un lato, Moulthrop vede con favore l'apertura di tematiche, tradizionalmente di competenza della retorica e della filosofia del linguaggio, ai contributi della *computer science*; dall'altro, citando J. Baudrillard, mette in guardia contro le insidie che si celano dietro la volontà di creare simulazioni della realtà e mondi artificiali. Auspica, inoltre, che i letterati controllino più direttamente gli sviluppi tecnologici, in un rapporto franco e costruttivo con i teorici della informazione. Le idee che scaturirebbero da questo dialogo potrebbero trovare applicazione sia nelle tecnologie che nella pratica artistica. Ed infatti arte e tecnologia sono due mondi che, a detta di Moulthrop, nei prossimi anni tenderanno sempre più a convergere.

Michael Joyce, dal canto suo, vede negli ipertesti la nascita di una nuova forma letteraria. Suo è uno dei primi romanzi interattivi, *Afternoon, a story*, costruito con un sistema ipertestuale chiamato Storyspace, che ha realizzato insieme a Bolter e Smith. A Pittsburgh egli ha presentato la nuova versione di Afternoon, che è stato modificato nell'interfaccia e si avvale ora di alcune tecniche di Intelligenza Artificiale per aumentare il grado di interattività fra lettore e sistema testuale.

Mentre gli ipertesti sono considerati generalmente strumenti di esplorazione di testi già esistenti, Joyce li vede, invece, come strumenti per la produzione di letteratura. In sostanza, egli afferma che la documentazione tecnica è governata dal mito dell'ordine emergente, ovvero dal fatto che la struttura del significato emerge dal testo. Il significato in letteratura emerge, invece, dalla interazione fra testo e lettura: il lettore è considerato a pieno titolo co-autore del testo. Gli autori della "narrativa ipertestuale" sono quindi cantastorie virtuali: la narrativa non è più tramandata irreversibilmente dallo scrittore al lettore, in quanto esiste un meccanismo retroattivo che, interpretando le risposte del lettore, ne potenzia il ruolo attivo rispetto al testo. Gli ipertesti, ha concluso Joyce, sono veramente più vicini alla creazione di conoscenza piuttosto che alla conservazione del sapere in strutture preordinate.

Alla discussione ha partecipato anche J. McDaid, l'ideatore di *Uncle Buddy's Phantom Funhouse*, un altro esempio di *interactive fiction-in-progress*. Gli ipertesti, secondo McDaid sono interfacce verso applicazioni che non hanno uno scopo preciso, fuorchè quello di liberare le possibilità di associazione del lettore.

Attività collaterali

Con il nome di *Birds of a feather* si sono tenute alcune riunioni informali, organizzate da chiunque volesse discutere su di un argomento specifico. Gli incontri più attesi sono stati quelli indetti da Douglas Engelbart e Ted Nelson ed il carattere dei due personaggi ha dato a questi incontri un taglio assai diverso.

Con legittimo orgoglio, Doug Engelbart ha raccontato le più significative esperienze della sua collaborazione con la McDonnel Douglas. Ormai al di sopra delle parti, ha sollecitato la collaborazione delle maggiori imprese del settore al progetto Bootstrap, che, secondo il suo ideatore, diverrà la più potente realizzazione nell'ambito del *groupware*.

Una grande carica di entusiasmo è venuta dall'incontro con Ted Nelson, ben noto "guru" degli ipertesti. In un monologo degno di un attore professionista ma sempre sapientemente dosato con autoironia, Ted Nelson ha lanciato il suo Developer Program e ha raccontato in breve del futuro di Xanadu. In sostanza, la Xanadu Operating Company, oggi sussidiaria alla Autodesk Inc., entro la metà dell'anno approntera un sever per workstation Sun e, più tardi, anche per Macintosh e IBM

compatibili. Il programma presentato intendeva promuovere lo sviluppo di applicazioni front-end per Xanadu su queste piattaforme iniziali.

Di un certo interesse sono state le dimostrazioni serali dedicate ad una quindicina di tool ipertestuali.

Degno di menzione l'esperimento di traduzione di un documento Notecards in formato Interchange e da questo formato in Hypercard. Il prototipo, ancora scarno ed essenziale, mostrava dal vivo alcune delle difficoltà che si presentano a chi vuole tentare di rendere i sistemi ipertesto convertibili fra loro, ma è parso certamente un ottimo inizio.

Molto interessata è sembrata anche un'interfaccia ipertestuale a Usenet, il servizio di notizie ed articoli online per utenti Unix. Il prototipo, sviluppato in ambiente Macintosh, consente la creazione automatica di link all'interno dei documenti presenti in Usenet, grazie ad un algoritmo basato sulla similarità fra stringhe di caratteri.

Abbiamo avuto occasione di vedere da vicino il famoso Perseus Project, sistema multimediale sulla letteratura, la storia e l'archeologia della Grecia Antica, sviluppato all'Università di Harvard, che, dopo due anni di ricerche, sta per essere completato. Il progetto Perseus collega, mediante link ipertestuali, testi dei maggiori autori greci con le relative traduzioni, immagini a colori di reperti archeologici (memorizzate su di un videodisco) e mappe della Grecia. L'utente può memorizzare il proprio cammino all'interno di questo grande archivio e prendere appunti su ciò che viene letto. Perseus è stato sviluppato in HyperCard e diventerà, all'inizio del 90, un Cd-Rom acquistabile per soli 50 dollari.

Interessanti sono sembrate anche le ultime novità di Intermedia, tra cui segnaliamo InterNoter, uno strumento per il lavoro d'equipe, che permette di fare annotazioni e di mantenere tutte le successive versioni dei documenti. Il sistema fornisce un nuovo paradigma di browsing che permette all'utilizzatore non solo di attraversare i link, ma anche di trasferire dati grazie ad essi.

Ricordiamo inoltre gIbis, lo strumento di supporto alle decisioni e di organizzazione di conoscenza per più utenti realizzato da J. Conklin su Germ, un'ambiente integrato che offre funzionalità di browsing (secondo il metodo entity-relationship), di editing grafico, oltre che di creazione di link.

L'Institute for Research and Learning di Palo Alto ha mostrato, infine, uno strumento per la catalogazione e il retrieval di video clip e filmati. Ideale per gli psicologi ed i ricercatori nell'ambito delle scienze sociali che utilizzano i filmati nell'analisi dei comportamenti, VideoNoter consente, inoltre, di associare ad ogni filmato commenti ed annotazioni, sia di tipo testuale che di tipo grafico.

PRIMO CONVEGNO DELL'ASSOCIAZIONE ITALIANA DI INTELLIGENZA ARTIFICIALE

Giulio Vian
Dipartimento di Scienze dell'Informazione
Università di Milano

Dall'8 al 10 novembre scorso si è tenuto presso la città di Trento il primo Convegno Nazionale dell'Associazione Italiana di Intelligenza Artificiale (AIIA), organizzato dall'Istituto Trentino di Cultura (ITC) e dall'Istituto per la Ricerca Scientifica e Tecnologica (IRST) e aperto ad un pubblico che ha annoverato circa 500 presenze. Questo successo in termini di afflusso per una disciplina che è una delle più giovani tra le varie branche della scienza degli elaboratori testimonia il fatto che, in Italia, gli interessi intorno all'Intelligenza Artificiale può ormai vantare un sostanzioso gruppo di addetti ai lavori e di interessati alle sue attività. La manifestazione ha condiviso due momenti: uno, classico, legato alla presentazione di lavori scientifici (42 selezionati su 104 giunti al comitato scientifico per la revisione) che hanno permesso di fare il punto su molte

attività e molti risultati che nel nostro Paese caratterizzano gli interessi di molte accademie e di centri di ricerca pubblici e privati, ed uno prettamente espositivo, nel quale i produttori e i commercianti di prodotti per l'Intelligenza Artificiale hanno potuto incontrare un pubblico selezionato.

La parte dedicata alla presentazione di lavori si è articolata secondo una suddivisione molto tipica delle varie aree che compongono il campo d'interesse coperto dall'Intelligenza Artificiale: rappresentazione della conoscenza, elaborazione del linguaggio naturale, visione e robotica, sistemi basati sulla conoscenza, ragionamento automatico, risoluzione di problemi e architetture e apprendimento automatico.

Per quel che riguarda la rappresentazione della conoscenza, ricordiamo un interessante intervento di C. Sossai e S. Badaloni, che hanno presentato un approccio al ragionamento temporale che prende origine dal lavoro di Shoham allargato a caratterizzazioni che permettono sia una rappresentazione esplicita delle conoscenze temporali, sia il ragionamento non monotono, sia il trattamento dell'incertezza.

Nel settore dell'elaborazione del linguaggio naturale, i lavori presentati sono stati caratterizzati da due indirizzi: le applicazioni pratiche e i problemi teorici ancora legati all'interpretazione in condizioni anomale. In questo contesto ricordiamo il lavoro di C. Ferrari et al. che ha evidenziato le potenzialità della comunicazione multimediale, con i problemi linguistici allargati al momento dell'elaborazione del linguaggio naturale e della gestualità.

La sezione dedicata ai sistemi basati sulla conoscenza ha visto in particolare il lavoro di Lanzola, Stefanelli, Barosi e Magnani, in cui una originale impostazione metodologica ha introdotto un modello diagnostico di indubbia originalità: il coinvolgimento del livello epistemologico nel modello proposto ha infatti permesso di mettere in luce il ruolo dell'abduzione nel ragionamento operato dall'esperto di manutenzione di sistemi (impiantista, medico) nella risoluzione di problemi. Tale lavoro ha permesso di offrire al pubblico un approfondimento dei suoi contenuti grazie all'esposizione di un altro lavoro di Ramoni, Stefanelli e Magnani, in cui la parte più formale del modello diagnostico proposto è stata presentata nel lavoro "Una teoria formale del ragionamento diagnostico". In questa sede viene infatti approfondito il particolare schema inferenziale fornito dall'abduzione per mezzo della sua modellazione in termini di ragionamento non monotonico.

Sempre in questa sezione il lavoro presentato da Brajnik, Chittaro, Costantini, Guida, Tasso e Toppano ha fornito interessanti spunti di discussione. Esso ha illustrato un modello di ragionamento su più livelli di profondità della conoscenza che spazia dalle più immediate associazioni fino ai modelli di sistemi fisici rappresentati in termini qualitativi e legati al ragionamento di senso comune.

Oltre alla presentazione di lavori da parte degli studiosi del nostro Paese, il Convegno di Trento ha rappresentato l'occasione per l'invito ad autorità internazionali della ricerca in Intelligenza Artificiale. Sul podio riservato agli inviti illustri si sono succeduti T. Poggio del prestigioso MIT, Y. Shoham e R. Engelmore ambedue dall'Università di Stanford. Le loro relazioni hanno messo in luce i più recenti risultati delle loro ricerche, in una esposizione abbastanza generale per poter presentare anche i possibili sviluppo futuri in senso teorico, applicativo e rispetto alle implicazioni epistemologiche e di crescita dell'Intelligenza Artificiale.

L'Associazione Italiana per l'Intelligenza Artificiale ha dimostrato, attraverso il successo del Convegno promosso e supportato, di raccogliere intorno a sè la parte più consistente della ricerca e delle applicazioni in questo settore a livello nazionale: mediante le innumerevoli iniziative che l'Associazione promuove (dalla redazione di un bollettino periodico contenente fatti, notizie e avvenimenti, all'organizzazione di Gruppi di Lavoro per settori specifici dell'Intelligenza Artificiale, fino alla sponsorizzazione di manifestazioni internazionali mediante premi e patrocinii) conferma il suo impegno a farsi portavoce di chi crede nella crescita di questa disciplina e nel contributo che il nostro Paese può fornire in questa direzione.

MODELLI SINCRONI, ASINCRONI E CONNESSIONISTI: UN CONVEGNO AICA SULLA PROGRAMMAZIONE PARALLELA

Andrea Vitali
Dipartimento di Scienze dell' Informazione
Università di Milano

La richiesta di elaboratori sempre piu potenti per la simulazione di modelli, da parte di comunità eterogenee finalizzate all'uso e allo sviluppo di hardware e di software, ha fatto sì che venisse incrementato l'impiego di supercomputer, definiti come gli elaboratori più potenti disponibili in un determinato arco di tempo, dove "potente" è inteso in termini di velocità di esecuzione, capacità di memoria e precisione di macchina. Ampi riferimenti storici testimoniano come il raggiungimento di queste potenze sia in parte dovuto all'uso di strumenti (hw/sw) sempre piu sofisticati e finalizzati allo sviluppo di forme di parallelismo (i\o, multiprogrammazione, memoria virtuale, cache, compilatori, etc).

Il Convegno Internazionale organizzato dall'AICA sulla Programmazione Parallela tenutosi al Cineca dal 23 al 25 dello scorso ottobre è stato un momento di dibattito tra vari esponenti di Università e Centri di Ricerca, finalizzato a fornire delle indicazioni sullo stato attuale dei modelli di programmazione sincrona, asincrona e connessionista. La suddivisione in base alle tematiche, oltre a rapportarsi a passi di generalizzazione crescente in un contesto spazio-tempo, sottolinea la volontà di usare linguaggi, contenenti intrinsecamente metodologie parallele e non forzature di strutture inerentemente sequenziali, facendo riferimento a vari argomenti come i modelli computazionali paralleli, algoritmi, modelli di elaborazione parallela e distribuita.

L'intervento introduttivo affidato al coordinatore N.Cesa Bianchi ha sottolineato come l'arbitrarietà della suddivisione, data dalla varietà delle tematiche interessate, cerchi di privilegiare gli aspetti software legati alla modellistica e alla linguistica. Il concetto di funzionalità ottima, avente dei riferimenti di base rispetto all' hardware usato, deve essere superato e arricchito da modelli che permettano di trovare algoritmi ottimali privilegiando l'aspetto linguistico rispetto ad un hardware considerato come variabile indipendente: non si vuole un unico modello ma i domini di eccellenza di ogni singolo modello. Assunto che la programmazione è la prova di correttezza del programma stesso si vogliono dei linguaggi tradizionali, arricchiti di strutture parallele innovative, per ambienti paralleli.

Il discorso del prof. Vanneschi ha focalizzato gli obiettivi del progetto finalizzato CNR ("Sistemi informatici e calcolo parallelo") e la contestualità di un sottoprogetto, dedicato all'elaborazione parallela, nei confronti di vari interventi delle sedi interessate. I punti trattati riguardavano anche le metodologie delle tecniche di progettazione e delle valutazioni delle prestazioni, la modellistica integrata e l'approccio emulativo basato su macchine esistenti come i transputer.

Le metodologie adottate per lo sviluppo del problema sono state ovviamente molteplici ma i risultati finora ottenuti non permettono di privilegiarne una in particolare, per la ragione già espressa e cioè mancanza di idee innovative nei confronti delle tecnologie esuberanti esistenti. Le architetture più promettenti analizzate sono state a nostro avviso le seguenti:

- Array processor: è un computer parallelo sincrono con più unita logico-aritmetiche chiamate PE (Processing Element) che può operare in modo SIMD (Single Instruction Multiple Data); permette un parallelismo di tipo spaziale ed è usato, per le sue caratteristiche funzionali, per il calcolo vettoriale.

- Pipeline processing : permette l'esecuzione di istruzioni successive in modo sovrapposto operando per mezzo di una architettura SISD (Single Instruction Single Data); permette inoltre un parallelismo di tipo temporale ed è usato, per le sue caratteristiche funzionali, per l'elaborazione scalare e per codice altamente iterativo.

-Cellular automata: sono sistemi dinamici dove lo spazio, il tempo e le variabili dinamiche sono discrete. La differenza che distingue i cellular automata dalle reti neuronali è la regolarità spaziale data dalla discretizzazione dello spazio; l'aggiornamento delle celle, basato su una architettura SIMD, è sincrono ed è funzione di una legge, a valori booleani, uguale per tutte le celle.

-Data flow machine: per definizione è un computer la cui computazione e controllata dal flusso di dati. La rete computazionale viene raffigurata con un grafo orientato per mezzo di archi sottesi dai vari nodi, implementativamente rappresentati dai moduli autonomi (tasks) computazionali. Le reti data flow possono essere data-driven (ogni processore elabora una computazione quando arrivano i suoi input e non esistono interazioni con gli altri processori) o demand-driven (ogni processore non effettua una computazione fino a quando non debba soddisfare una richiesta da un altro processore): l'overhead e dato runtime.

-Sistemi MIMD: è un'architettura multiprocessore (Multiple Instruction Multiple Data) con la peculiarita di favorire interazioni, tra gli n processori, per mezzo di flussi di memoria provenienti da uno stesso spazio dei dati condiviso da tutti i processori.

-Sistemi ad ipercubo: un ipercubo k-dimensionale è una architettura parallela composta da 2 elevato alla k nodi processori dove due nodi sono collegati direttamente solo se il loro numero identificatore in binario differisce esattamente di un bit: al più k-archi sono necessari per connettere 2 vertici arbitrari. La comunicazione tra processi concorrenti è basata usualmente su un meccanismo di sincronizzazione a scambio di messaggi (i nodi posseggono una memoria locale senza memoria condivisa tra processori); la tecnica di programmazione utilizzata si basa sulla replicazione di uno stesso algoritmo sequenziale su n processori, che risolveranno il problema su altrettanti insiemi distinti di dati estratti da una partizione del dominio originario.

-Neural network: può essere definita in modo bivalente, sia come un aggregato di automi a stati finiti sia come un sistema i cui parametri vengano modificati dinamicamente. Le caratteristiche, comuni ad entrambi i modelli, sono date dall'elevato numero di neuroni e di interconnessioni tra gli stessi e dalla presenza di operazioni elementari a livello di singoli neuroni.

La simulazione dei più grandi supercomputer commerciali tramite l'uso di modelli sincroni quali gli array processor e il pipeline processing, ha favorito l'evoluzione e l'uso di programmi eseguibili ottenuti mediante l'utilizzo, più o meno combinato, di compilatori "intelligenti" e di linguaggi di programmazione arricchiti di costrutti sintattici, tali da permettere certe operazioni simultaneamente su molti dati. Malgrado l'evidente vantaggio dato dalla possibilità di migliorare vecchi programmi con uno sforzo minimo, l'impiego di compilatori vettorizzanti è adeguato solo per certi tipi di software poichè consente un incremento limitato della velocità. I vantaggi più evidenti si avranno piuttosto nel software scientifico: i costrutti linguistici per il trattamento dei dati in forma matriciale, messi a disposizione dal Fortran o da linguaggi elaborati ad hoc ispirati al Fortran, che adottino un supporto statico facente uso di questi compilatori, beneficeranno di supporti atti a riconoscere e tradurre in modo efficiente le strutture vettoriali, a rimuovere eventuali ostacoli che inibiscono la vettorizzazione del codice e ad ottimizzare delle strutture vettoriali esplicitandole attraverso espansioni inline. L'impiego delle diverse architetture, in funzione delle loro modularità e regolarità, rifletterà quindi i diversi approcci adottati. A titolo esemplificativo si può dire che gli array processor ottimizzeranno il trattamento di codice matriciale mentre il pipelining guadagnerà in prestazioni se sfruttato tramite codice il più possibile lineare o con finalita facenti uso del concetto di microtasking (tecnica di parallelizzazione che individua il singolo Do-loop come entità di programma da suddividere in un certo numero di task da assegnare ai singoli processori virtuali). L'intervento a cura degli ospiti del Cineca ha sottolineato l'analisi evolutiva dell' elaborazione vettoriale ponendo l'accento sulla necessità da parte della comunità scientifica di attuare concretamente un cambiamento a favore della programmazione vettoriale e parallela.

Un altro linguaggio che ha subito una fase evolutiva è il Prolog da cui ha avuto origine il linguaggio logico concorrente Shared Prolog, analizzato in relazione ai modelli asincroni: l'idea fondamentale è quella di interpretare un obiettivo come un insieme di processi logici che comunicano attraverso scambi di messaggi attraverso una memoria di lavoro comune (blackboard), senza l'uso di condivise. In astratto, la computazione di un obiettivo logico può essere vista come un albero dove la radice è l'obiettivo iniziale mentre gli altri nodi sono marcati dai task derivati: la soluzione è data dalla regola inferenziale. I due tipi di parallelismo cui si fa ricorso per l'esecuzione di un programma Prolog sono l'or-parallelism (esame simultaneo di diverse condi-

zioni per un dato obiettivo non deterministico) e l'and-parallelism (esame sequenziale di diverse condizioni per un dato obiettivo deterministico): l'or-parallelism mette a disposizione del programmatore, grazie al parallelismo esplicito offerto dal linguaggio stesso, la possibilità di avere un pieno controllo sia sulla schedulazione dei processi che sulla granularita della comunicazione. Altri approcci sperimentati sono quelli riguardanti il linguaggio ECSP e il sistema DISC. ECSP è un linguaggio di sistema concorrente di alto livello finalizzato, tramite un'architettura multi-transputer, al supporto efficiente ed affidabile di linguaggi applicativi di alto livello ed al design di un sistema operativo decentralizzato. Il sistema DISC e un ambiente a memoria distribuita per lo sviluppo, indipendente dalla architettura fisica della macchina, di software parallelo. Esso comprende un linguaggio concorrente, un insieme di strumenti di supporto allo sviluppo di programmi e all'esecuzione del linguaggio. La tecnica usata più comunemente nelle macchine di tipo MIMD, a memoria distribuita, è basata sull'integrazione di linguaggi tradizionali, imperativi e non, con opportune biblioteche di sottoprogrammi per la gestione del parallelismo.

Secondo la relazione del prof. Golshani, il ricorso alle data flow machine, applicabili a sistemi esperti/basati sulla conoscenza, consente un doppio livello di parallelismo dato sia a livello di memorizzazione, facendo uso di moderni e sofisticati devices come le memorie associative, processo computazionale globale; ogni processore del sistema è visto come una database machine. La proposta di un modello, descritto dal prof. Misra, che permetta una effettiva manipolazione delle specifiche, del non determinismo e della non terminazione dovrebbe fare uso di una metodologia chiamata Unity: essa permette una programmazione parallela e un approccio formale, attraverso delle regole di inferenza, per dedurre proprietà dalle specifiche e provare la correttezza delle implementazioni possibili.

Le tematiche connesse allo studio e alle applicazioni delle reti neuronali sono state illustrate dal prof. Apolloni e dal prof. Mauri, entrambi concordi nell'affermare come attualmente si siano sviluppate, oltre all' uso di strumenti software/hardware per la simulazione di reti di neuroni ed alle numerose attivita di ricerca, applicazioni a problemi concreti quali la diagnosi dell'epilessia ed il controllo ad alta precisione di un braccio flessibile. Ulteriori progressi sono stati riscontrati nell'uso delle reti neuronali come modello computazionale per la soluzione di problemi di ottimizzazione e di riconoscimento di forme e per l'implementazione di memorie associative.

Il processo conoscitivo è fondato sul concetto di educazione inteso come costruzione graduale di una routine (non ancora disponibile nelle librerie software) suffragata dagli esempi di come la routine stessa debba comportarsi. Il fattore che condiziona maggiormente l'uso di queste routine è che la disponibilità di buoni algoritmi di apprendimento mediante reti di neuroni è legata alle particolarità offerte dallo sviluppo di metodi ad hoc per risolvere specifici problemi. Il fatto di non essere in grado di codificarla in un linguaggio parallelo altamente efficiente suggerisce l'ipotesi di affidare al computer stesso la scrittura del codice della routine richiesta, dopo aver definito come informarlo di quella routine e quale grado di errore si è disposti a tollerare.

L'epilogo del convegno è stato un momento di confronto tra i vari paradigmi di programmazione parallela. Poichè non esiste un unico approccio formale al parallelismo, si è proposto l'impiego di una semantica denotazionale che permetta l'utilizzo di nuovo software parallelo per adattarvi una architettura che ne espliciti il più possibile le potenzialità. La mancanza di linguaggi paralleli general purpose e di linguaggi paralleli contenenti istruzioni vettoriali ad alto livello ed efficienti ha certamente contribuito all'evoluzione sia di linguaggi imperativi, adoperanti la comunicazione punto punto tra processi con parallelismo esplicitabile a livello di programmatore, sia di linguaggi logici e funzionali ad alto costo computazionale, facenti uso di un elevato formalismo matematico a parallelismo implicito. In ogni caso si è sottolineata la presenza di troppi ordini di grandezza tra parallelismo e sequenzialità; il relativo divario è parzialmente recuperato mediante l'utilizzo, non sempre possibile, di un compilatore vettorizzante, per la traduzione di programmi sequenziali in programmi paralleli.

In conclusione questo convegno sulla programmazione parallela ha sicuramente offerto delle risposte ai vari problemi analizzati, ma bisogna precisare come attualmente non vi sia ancora una coscienza comune tale da fare intravedere, a medio-lungo termine, uno sviluppo armonico e organico di una problematica così complessa e cruciale quale è la programmazione parallela.

RECENSIONI

J. Berstel, C. Reutenauer, RATIONAL SERIES AND THEIR LANGUAGES EATCS, Monographs on Theoretical Computer Science 12, Springer-Verlag, 1988 151 pp.

Il concetto di serie formale riveste un'importanza fondamentale per la trattazione uniforme di un ampio numero di problemi in ambito combinatorio. Un problema espresso in termini di serie formali può essere quindi soggetto all'uso di strumenti analitici e algebrici operanti direttamente nello spazio di tali serie. In particolare, due sono le aree in cui le serie formali sono di frequente utilizzo, e che riguardano i problemi di conteggio (per quanto riguarda l'applicazione strettamente legata alla matematica combinatoria) e quelli di classificazione di linguaggi (studio dell'ambiguità nella teoria dei linguaggi formali).
Questo libro vuol essere un'introduzione alle serie formali razionali a variabili non commutative e alle relazioni esistenti tra queste e i linguaggi formali e la teoria dei codici.
Il contenuto del volume è logicamente organizzato in tre parti.
La prima parte introduce i concetti di serie formale razionale e di serie riconoscibile: essa contiene i risultati di tipo generale più importanti riguardanti ad esempio l'equivalenza tra serie razionali e serie riconoscibili (teorema di Schuetzenberger) e la costruzione di rappresentazioni lineari ridotte per le serie riconoscibili.
La seconda parte affronta diversi temi. Dapprima viene studiato il rapporto tra serie razionali e linguaggi formali (teorema di Kleene), quindi viene analizzato il caso relativo a serie formali razionali in una sola variabile illustrando le proprietà aritmetiche di tali serie e le relazioni esistenti rispetto alla natura dei coefficienti (serie a coefficienti interi positivi).
Tra gli argomenti trattati nell'ultima parte troviamo un capitolo dedicato ai problemi di decisione riguardanti i supporti di serie razionali: i risultati forniti sono relativi a problemi decidibili.
Infine, vengono studiate le proprietà algebriche dei polinomi a variabili non commutative, e analizzate le relazioni tra questi e la teoria dei codici.
Complessivamente si tratta quindi di un'opera specialistica che analizza il tema delle serie formali razionali adottando un'approccio esclusivamente di tipo algebrico. I risultati classici appartenenti a tale area sono chiaramente esposti, soprattutto per quanto riguarda le correlazioni con la teoria dei linguaggi. Apprezzabile è anche l'analisi delle proprietà aritmetiche di particolari famiglie di serie razionali. La visione globale che ne scaturisce è sufficientemente omogenea. Alcune piccole imprecisioni presenti sono conseguenza più che altro dello stile sintetico adottato.

Paolo Massazza

J. Ullman, PRINCIPLES OF DATABASE AND KNOWLEDGE-BASE SYSTEMS, Computer Science Press, 1988, 2 Voll.

Appartiene a quella categoria di libri destinati a costituire un punto di riferimento d'obbligo per tutti coloro, studenti o operatori nel settore delle basi di dati non proprio alle prime armi, che intendono approfondire le tematiche connesse all'area del *database design* con uno sguardo alle prospettive future fornite da un'analisi dettagliata, e nello stesso tempo globale, di una persona con esperienza pluriennale in questo campo.

Jeffrey Ullman è noto, negli ambienti accademici e non, come interprete e principale attore di tutte le trasformazioni in atto nel poliedrico settore dei *database system*, ed è autore di un altro famosissimo libro, PRINCIPLES OF DATABASE SYSTEMS (1982), di cui quello del 1988 vuole essere una sostituzione ed una estensione. Redatto in due volumi, PRINCIPLES OF DATABASE AND KNOWLEDGE-BASE SYSTEMS non intende essere un libro sulla rappresentazione della conoscenza (gli addetti al settore dell'intelligenza artificiale avranno ben poco da imparare da questo libro, visto che viene fatta una introduzione ai linguaggi logici), quanto piuttosto un testo sulla filosofia della rappresentazione dei dati e un preciso ed esauriente trattato sulle tematiche connesse in vario modo alla progettazione di database, frutto delle conoscenze, ripensate in chiave sistematica, provenienti dal vasto background dell'autore.

L'esposizione spazia dagli argomenti base del settore, quali i modelli dei dati, i modelli logici, l'organizzazione fisica dei dati, i database ralazionali, i problemi di sicurezza ed integrità dei dati, agli argomenti più stimolanti, perchè ancora avvolti dal fascino dell'inesplorato, dei database object-oriented, o del controllo di processi concorrenti che accedono allo stesso database, con relativa gestione delle transazioni.

Non è trascurato anche il profilo più pratico di queste ultime innovative proposte: in particolare, per ciò che riguarda il capitolo dedicato all'approccio object-oriented, viene presentato un esempio di creazione e manipolazione di database tramite il linguaggio ad oggetti OPAL, mentre per il capitolo relativo al controllo della concorrenza sono illustrate tutte le variazioni sull'uso della tecnica dei protocolli a due e tre fasi. Le tematiche più classiche, come le modalità di organizzazione fisica e logica dei dati e le tecniche di progettazione di DB relazionali, sono messe in adeguato rilievo; vengono infatti trattate esaurientemente le strutture dati più comuni dei database come i file indicizzati, hash file e hash function, i B-tree e le rappresentazioni delle dipendenze funzionali, ecc.
Anche al problema della integrità e sicurezza dei dati viene riservato largo spazio, così come alla ''prevenzione'' e alla ''cura'' dei deadlock.

La caratteristica peculiare dell'opera, comunque, è quella di riuscire ad inserire i classici paradigmi della rappresentazione object-oriented nella trattazione dei problemi sopra accennati. Fra gli aspetti più interessanti è senz'altro da sottolineare l'importanza attribuita a certi aspetti legati alle problematiche dell'informatica distribuita: la presenza di database situati fisicamente su più server, connesse fra loro tramite LAN, costituisce a tutt'oggi una prospettiva che attrae per i potenziali benefici ma che lascia intuire anche la complessa trama di difficoltà (di gestione globale dell'architettura) che immediatamente insorgono.

Dopo aver illustrato nel primo volume i fondamenti della teoria dei database, il secondo volume del libro costringe l'attenzione del lettore a focalizzarsi sui risvolti pragmatici dei sistemi knowledge-base; lo scopo è quello di mostrare come un corpo di nozioni teoriche possano essere velocemente trasferite nel settore applicativo, rendendo disponibili a breve termine gli strumenti di cui si è ampliamente parlato nel volume precedente.

Per questo motivo, Ullman propone una vasta trattazione degli algoritmi di ottimizzazione delle queries, compreso il modello ''relazione universale'' per la comprensione di queries poste in linguaggio naturale; la trattazione dei diversi algoritmi interessa l'intero secondo volume e costituisce in sostanza la parte più ''tecnica'' e forse meno avvincente del libro. Come l'autore stesso evidenzia, a tutt'oggi (88-89), non sono ancora disponibili database realmente basati sulla conoscenza; nonostante ciò, Ullman opera una approfondita analisi di alcuni progetti in corso (NAIL!, coordinato da Ullman stesso, LDL, POSTQUEL) che vogliono essere l'esemplificazione implementativa dei concetti più importanti espressi nel libro. L'insieme risultante è comunque notevole per la compattezza e la chiarezza espositiva, nonchè per la vastità delle tematiche e la loro esauriente trattazione, e ha tutti i numeri per riuscire il best seller del mondo dei database per i prossimi anni.

Alberto Fioravanti

Ernest R. Tello, OBJECT ORIENTED PROGRAMMING FOR ARTIFICIAL INTELLIGENCE: A guide to Tools and System Design, Addison-Wesley, 1989, 335 pp.

Si tratta di un libro orientato a coinvolgere operatori dell'Intelligenza Artificiale (AI) sia del settore accademico che commerciale, lasciando, alla fine, forse un po' più scontenti i primi e un po' più contenti i secondi. È particolarmente adatto per essere un complemento ad un qualsiasi tool object oriented utilizzabile su macchine a basso costo, tipicamente su PC.

L'attenzione maggiore viene posta nello scegliere il tool giusto per il progetto giusto e nello stimolare l'attività di programmazione, oggi emergente più che mai, orientata agli oggetti, mostrando come sia possibile applicarla ad alcuni problemi di AI sia di carattere accademico che commerciale.

Da una prima introduzione che si preoccupa di precisare i principali concetti di OOP (Object Oriented Programming) per i neofiti, il testo si snoda esponendo, sempre in modo molto divulgativo, quali possono essere

i vantaggi dell'utilizzo delle tecniche OOP in AI, che vengono successivamente esemplificati, facendo riferimento ad alcune recenti applicazioni AI: FORMES, STARPLAN II, PRIDE, CYC e ROOMS, l'ultima idea dei laboratori Xerox, che estende il concetto fondamentale dell'interfaccia utente dalla "finestra", alla "stanza".

Dopo una articolata descrizione dei tool strettamente orientati alla OOP, nella quale Smalltalk e C++ la fanno da padroni, vengono passati in rassegna altri strumenti OOP più orientati verso l'intelligenza artificiale, che possono essere considerati come delle estensioni del LISP. in particolare CLOS, che è una estensione object oriented del Common Lisp.

Vengono utilizzate anche, ed è probabilmente la parte più interessante del testo per chi tratta AI da un punto di vista commerciale, le principali shell per sistemi esperti. Goldworks II, ART, LOOPS, KEE vengono presentate in successione, mettendone in evidenza le strutture fondamentali, le ambizioni e le possibilità. Nexpert Object, una shell che si sta diffondendo ovunque con notevole rapidità, viene utilizzata nel capitolo successivo per esemplificare una suggestiva applicazione, consiste nella diagnosi di malesseri dovuti all'assenza di gravità.

Infine le ultime pagine del testo esplorano dapprima alcune tecniche object oriented per la rappresentazione di problemi reali, in particolare la possibilità di operare forme di simulazione continua o discreta basata sulle regole, ed introducono sommariamente un argomento che attualmente riscuote particolare interesse nella comunità AL d'oltreoceano, cioè le reti neuronali; successivamente vengono esaminati alcuni tool che permettono la gestione della concorrenza dei processi.

Se da una parte bisogna riconoscere all'autore di aver realizzato un'opera caratterizzata da un forte senso pratico, di aver esposto i contenuti con la massima chiarezza, di aver messo ordine nel caos di denominazioni tecnicistiche, identificando concetti uguali in contesti diversi, di aver fatto riferimento a prodotti e situazioni estremamente aggiornati, fatto che rende l'opera quasi uno stato dell'arte dell'OOP e dell'AI, dall'altra gli si può rimproverare di aver trascurato forse eccessivamente le basi teoriche dell'OOP e dell'AI, di aver dato un'impronta alquanto "introduttiva" al contenuto, di aver utilizzato spesso esempi troppo classici di utilizzo dei tool; ad esempio nell'analisi delle shell vengono spesso illustrate le demo vendute con il prodotto stesso.

Inoltre l'esposizione delle caratteristiche di un tool risultano talvolta troppo simili a quelle illustrate nel manuale del tool stesso, e risultano piuttosto acritiche.

Si tratta, insomme, di un libro utile per chi vuole introdursi nell'OOP dalla porta centrale e nell'AI da una porta secondaria; inoltre può essere considerato un prezioso compendio per chi vuole avere una visione sufficientemente dettagliata di tutti i principali tool per realizzare OOP ed AI, in particolare per chi ha il problema di scegliere lo strumento più adatto per sviluppare un determinato progetto; non è altrettanto utile, invece, per chi è già inserito nell'OOP o nell'AI e ha il problema di conoscere approfonditamente il proprio strumento di sviluppo.

Alberto Pozzi

J.A. Anderson e E. Rosenfeld (eds.), NEUROCOMPUTING: FOUNDATIONS OF RESEARCH, MIT Press, Cambridge, MA, 1988, pp. 729, ISBN 0 - 262 - 01097 - 6.

Dopo aver lanciato la new wave del Connessionismo anni '80 con i due volumi (ormai mitici) "PDP: Explorations in the Microstructure of Cognition", la MIT Press - a due anni di distanza - ci offre questa raccolta di lavori che formano un corposo quadro storico dello sviluppo del Neurocomputing dagli albori ai giorni nostri: l'intento dell'opera è infatti quello di tracciare una storia di questa disciplina attraverso 43 articoli, dal 1890 al 1987, aperti da una breve introduzione che fornisce le basi storico-scientifiche utili alla comprensione del testo.

Se un paio di anni fa Tommaso Poggio paragonava ironicamente il Connessionismo ad un virus che infettava la comunità scientifica a scadenze ventennali, il volume curato da Anderson e Rosenfeld si batte per una riabilitazione mostrando addirittura come lo stesso von Neumann traesse ispirazione dal primo lavoro sulle reti neuronali-firmato nel '43 da McCulloch e Pitts e incluso nella raccolta-per lo sviluppo della teoria degli automi formali alle basi della moderna computer science.

Il taglio culturale dell'opera in parte ricalca quello cognitivista dei due volumi "PDP: Explorations..." pubblicati dalla stessa casa editrice. Non a caso l'articolo di apertura è dello psicologo americano W. James (1890) e contiene profetiche intuizioni sull'attuale modellistica cerebrale basata sulle reti neuronali. Più avanti troviamo un altro "milestone" connessionista firmato da D.O. Hebb (1949) dove si illustrano i primi germi di una teoria computazionale dell'apprendimento. L'articolo successivo è il famoso "In Search of the Engram" di K.Lashley il cui sottotitolo (non riportato) "thirty years of frustration" lascia intendere quale sia stato l'esito di questa ricerca. I risultati di Lashley vengono spesso utilizzati dai fautori della cosiddetta "ipotesi distribuita" dei pattern cerebrali per sconfessare i "localizionisti", i quali appunto credono all'esistenza di aree iperspecializzate nel cervello. I cognitivisti americani (soprattutto quelli californiani) sostengono maggiormente l'ipotesi distribuita sulla quale si basa il paradigma computazionale noto come "Parallel Distributed Processing" (PDP). Comunque le attuali tendenze, come i più recenti risultati in neurofisiologia sembrano confermare, sembrano condurre verso un'ipotesi "ibrida" che contempla componenti sia localizzate che distribuite.

I lavori che coprono gli anni '50 e '60 sono soprattutto incentrati sul Perceptrone di Rosenblatt e sulle sue potenzialità. Questo algoritmo ebbe un notevole successo in quanto rappresentò il primo modello computazionale di apprendimento (basato su reti neuronali) applicabile in una quantità di casi, soprattutto nel campo della visione. Il termine di questa prima fase di entusiasmo è segnato da Minsky e Papert che nel '69 pubblicano "Perceptrons" (incluso nella raccolta) nel quale i limiti dell'algoritmo di Rosenblatt vengono pesantemente evidenziati.

Gli anni '70 furono un po' il Medioevo del Connessionismo, gli animi sfiduciati dalla "stroncatura" dei Perceptroni e i finanziamenti carenti fecero calare il numero di ricercatori e solo una minoranza di grossi nomi continuò a lavorare nel campo. Fra i lavori riportati dalla raccolta e relativi a questo periodo possiamo citare Cooper: "A Possible Organization of Animal Memory and Learning", Grossberg: "Adaptive Pattern Classification and Universal Recoding", Kohonen: "Correlation Matrix Memories", Marr e Poggio: "Cooperative Computation of Stereo Disparity", Amari: "Neural Theory of Association and Concept-formation".

Per quanto riguarda l'inizio degli anni '80, gli anni del boom del neurocomputing, sono riportati -fra gli altri- i contributi di Hopfield: "Neural Networks and Physical Systems with Emergent Collective Computational Abilities" (1982), Feldman: " Connectionist Models and their Properties" (1982) e di Fukushima col suo Neocognitrone (1983).

Fra i restanti articoli compresi nel volume c'è il lavoro di Ackley, Hinton e Sejnowsky che introduce le Boltzmann Machines (1985) e quello di Rumelhart, Hinton e Williams (1986) che illustra l'algoritmo di Backpropagation per Perceptroni multistrato; con quest'ultimo lavoro si dimostra l'infondatezza delle speculazioni di Minsky e Papert sulle limitazioni *intrinseche* del Perceptrone che, opportunamente generalizzato, diventa uno strumento assai flessibile per l'approssimazione di funzioni arbitrarie. Per ciò che concerne le realizzazioni hardware di reti neuronali soltanto due esempi sono citati: l'implementazione ottica del modello di Hopfield dovuta a Farhat, Psaltis, Prata e Paek e e un'implementazione VLSI riportata da Salviotti, Mahgowald e Mead.

Un'opera di questo genere, a parte le inevtabili parzialità nella scelta degli articoli riportati, è senza dubbio utile ad ampliare le prospettive storiche del ricercatore fornendo un quadro ad ampio respiro della materia con una discreta attenzione verso l'interdisciplinarietà.

Nicolò Cesa-Bianchi

G.S. Almasi, A. Gottlieb, HIGHLY PARALLEL COMPUTING, Benjamin/Cummings, Redwood City (CA), 1989, pp. xxiv+520, ISBN 0-8053-0177-1.

Credo che tanto l'autore quanto l'editore di un libro abbiano raggiunto il loro obiettivo quando il potenziale lettore, sfogliando il volume, prova il desiderio di leggerlo. Non è così scontato: molti libri si comprano per dovere, o per la pressione della pubblicità, o per fiducia nell'autore... Pensate a uno studente di fronte a un trattato di 500 pagine: sta male al solo pensiero di doverlo leggere e imparare!

Il libro che recensisco oggi a me ha fatto subito una buona impressione: sfogliandolo ci si trova di fronte quasi a ogni pagina a una figura, a una tabella o a un riquadro che colgono l'attenzione, incuriosiscono e

fanno venir voglia di leggere. Che cosa sarà mai "la follia di Dennard" (pag.100)? Perchè a pag. 13 c'è un'illustrazione che rappresenta dei leoni a caccia di bufali? Qual è "l'importanza di essere atomico" (pag. 260)? Non so nulla di Prolog, poniamo, ma se a pag. 184 trovo un riquadro sulla "terminologia di Prolog" probabilmente non ho molto da temere. Via - direte voi - chi non sa nulla di Prolog? E come ve la cavate con le equazioni di Navier-Stokes (pag.34)? Sono fondamentali per la simulazione dell'atmosfera nella previsione del tempo, un campo tipico di applicazione dell'elaborazione parallela.

Il guaio, parlando di parallelismo, è che si spazia dall'hardware al software, dai sistemi operativi agli algoritmi, dai database ai circuiti VLSI, dalle reti ai linguaggi, dal connessionismo ai compilatori... Certo non si può approfondire tutto, ma non si può neanche parlare solo di hardware ignorando le difficoltà di individuare adeguati algoritmi paralleli, e solo una storia dei principali sistemi paralleli finora realizzati, con quello che tali esperienze hanno insegnato, occupa non poche pagine.

Almasi e Gottlieb, il primo del centro di ricerca IBM T.J. Watson e il secondo docente del Courant Institute, hanno scelto di presentare una panoramica ampia dei diversi aspetti dell'elaborazione parallela, soffermandosi però quasi esclusivamente sul parallelismo elevato, su quel parallelismo, cioè, che dovrebbe rendere più veloce l'elaborazione di un unico, grande problema, piuttosto che incrementare il numero di lavori diversi che un elaboratore può svolgere in un dato tempo. Pur con questa limitazione, gli argomenti da trattare sono tanti da richiedere una presentazione generale di orientamento del lettore, che molto opportunamente viene fatta nel primo capitolo. La trattazione è comunque molto ordinata e ben suddivisa in una prima parte sui fondamenti, una seconda sul software e una terza sulle architetture.

Naturalmente il lettore è libero di scegliere l'itinerario di lettura che preferisce, ma, per esempio, la scelta degli autori di riservare il secondo capitolo alle applicazioni, rovesciando l'ordine tradizionale che relega le applicazioni alla fine, è significativa dell'intento di motivare il lettore e di renderlo consapevole delle problematiche prima che delle possibili soluzioni.

Trovo curioso che molti testi che trattano, principalmente o marginalmente, dell'architettura degli elaboratori (paralleli e non) non indicano nulla sulle tecnologie: mi chiedo se gli autori diano per scontato che il lettore sappia tutto, o ritengano irrilevanti tali aspetti, o non si sentano in grado di parlarne. Nel caso del nostro volume, il terzo capitolo presenta una rapida rassegna (30 pagine) sulla tecnologia dei circuiti e delle memorie.

Il quarto capitolo conclude la prima parte sui fondamenti, e riguarda modelli e algoritmi del calcolo parallelo. Si parla naturalmente della classificazione di Flynn (SIMD e MIMD) e successive variazioni, ma anche di complessità di calcolo, di PRAM e MP-RAM, e si esemplifica con l'analisi di tre algoritmi.

La parte dedicata al software è logicamente suddivisa in: linguaggi, compilatori e sistemi operativi. Dopo un esame dei costrutti essenziali per l'elaborazione parallela si passa a considerare i diversi linguaggi, sia quelli tradizionali con estensioni per l'elaborazione parallela sia quelli specifici. Gli autori mostrano come sia possibile raggruppare i linguaggi esistenti in otto categorie, e discutono vantaggi e svantaggi dei diversi approcci, dopo averli adeguatamente illustrati con esempi. Da rilevare l'assenza di ogni acceno al sistema di programmazione parallela "Linda": il lettore che volesse documentarsi in proposito potrebbe partire dall'articolo di D. Gelernter pubblicato su "Le Scienze" dell'ottobre 1989.

Per quanto riguarda compilatori e sistemi operativi, gli autori richiamano le nozioni di base, presentano gli aspetti aggiuntivi introdotti dal parallelismo, e tra l'altro rifanno la storia dei sistemi operativi paralleli, dal 1960 in poi.

Veniamo infine alla terza parte, quella dedicata alle architetture, che è certamente la più sostanziale (oltre 200 pagine), ma che è stata adeguatamente preparata da ciò che precede. Il primo capitolo riguarda l'interconnessione delle diverse unità di elaborazione (com'è noto, esistono almeno una dozzina di tipologie d'interconnessione statiche, più quelle dinamiche).

Gli altri capitoli esaminano rispettivamente le architetture SIMD, MIMD e ibride. Particolare spazio viene riservato ai sistemi CRAY, ILLIAC IV, GF11 e Connection Machine tra i SIMD, ma sono naturalmente i sistemi MIMD a fare la parte del leone, occupando oltre 120 pagine. La distinzione tra sistemi a memoria privata (o distribuita) e sistemi a memoria condivisa, introdotta sin dalle prime pagine, è qui ripresa e fornisce la linea principale della classificazione.

Tra i sistemi MIMD descritti con maggiore dettaglio vi sono naturalmente l'Ultracomputer della New York

University e l'RP3 IBM, i due progetti in cui gli autori sono direttamente coinvolti, ma l'equilibrio con la descrizione degli altri sistemi sembra sostanzialmente rispettato. Infine, tra i sistemi ibridi sono descritte le macchine con parole lunghe (VLIW) e i sistemi SIMD multipli.
Completano il testo l'elenco delle 145 figure, una bibliografia con 353 riferimenti (viene riportata anche la pagina in cui compare la prima citazione del riferimento, il che può esser utile) e un indice analitico esteso ma non accuratissimo: alcuni argomenti sono trattati anche in pagine diverse da quelle riportate. In conclusione, mi pare si tratti di un eccellente volume di introduzione ad ampio spettro all'elaborazione parallela, con il grosso pregio di un'ottima leggibilità.

Mauro Torelli

La redazione si scusa con gli autori e i lettori per i refusi comparsi nella rubrica "Recensioni" del N. 46 Dicembre 1989 di Note di Software dovuti a disguidi tecnici.

Errata Corrige

pagina 75 I^a colonna riga 3	*esplosiva / espositiva*
pagina 75 II^a colonna riga 2	*sistologici / sistolici*
pagina 75 II^a colonna righe 3-4	*Von Newmann / von Neumann*
pagina 76 I^a colonna righe 16-20	*Von Newmann / von Neumann*
pagina 76 I^a colonna riga 40	*a un livello sempre più accessibile / sempre a un livello più che accessibile*
pagina 76 I^a colonna riga 41	*alla stocastica / alla teoria stocastica*
pagina 76 II^a colonna riga 3	*probabilità delle / probabilità e delle*
pagina 76 II^a colonna riga 8	*spresso / spesso*
pagina 76 II^a colonna riga 19	*e nel libro "The Complexity of Computer" / e il libro di Savage "The Complexity of Computing"*
pagina 76 II^a colonna riga 26	*minizzazione / minimizzazione*
pagina 76 II^a colonna riga 44	*Edizioni 1989 / Edizioni Unicopli, 2^a edizione 1989*
pagina 79	*il nome dell'autore della recensione è Alessandro Malacart*

EDITO DA CALEIDOGRAF S.R.L.
VIA L. DA VINCI N.4/6 TEL. 039 - 587541
22058 OSNAGO (CO)

Spedizione in abbonamento postale, Gruppo IV, 1° semestre.
Pubblicità Inferiore al 70%
N. 36, 47, Marzo 1990
TAXE PERCUE